VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Použití balíčku TikZ pro sazbu LaTeXových knih s obrázky

Tikz Package for LaTeX Documents Typesetting

2013 Tomáš Pavlorek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Pavlorek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Použití balíčku TikZ pro sazbu LaTeXových knih s obrázky
Tikz Package for LaTeX Documents Typesetting

Zásady pro vypracování:

Cílem práce je popsat práci s balíčkem TikZ při sazbě obrázků systémem LaTeX. Práce se zaměří na typické příklady obrázků v elektrotechnické praxi - elektrotechnická schémata, bloková schémata, průběhy funkcí, grafy naměřených hodnot.

1. Popis struktury práce s obrázky v balíčku TikZ, pozicování - matice, řetězec.
2. Vytvoření vzorových obrázků s použitím defaultních bloků.
3. Tvorba nových bloků, jejich pozicování absolutní a relativní.
4. Ukázková sazba obrázků s elektrotechnickými schématy, blokovými schématy, vývojovými diagramy, logickými obvody, grafy naměřených hodnot a matematických funkcí.
5. Sazba pokročilých obrázků.
6. Animace v PDF s použitím TikZ.
7. Vytvoření webových stránek s prezentací výsledků práce.

Seznam doporučené odborné literatury:

[1] Rybička, Jiří. *LaTeX pro začátečníky*. 3. vyd. Brno: 2003, Konvoj. 238 stran. ISBN 80-7302-049-1
[2] Návod k balíčku TikZ, dostupný pod odkazem Documentation na URL: http://www.texample.net/tikz/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Na tomto místě bych chtěl poděkovat vedoucímu své bakalářské práce Ing. Janu Skapovi, Ph.D. za jeho odborné vedení, cenné připomínky a čas strávený při konzultacích této práce.

Abstrakt

Cílem bakalářské práce je popsat práci s balíčkem Ti*k*Z při sazbě obrázků systémem LaTeX. Práce je psána formou uživatelské příručky především pro začínající uživatele. Je rozdělena do několika kapitol v rámci nichž se zaměřuji na tvorbu obrázků v oblasti elektrotechniky a informatiky. Jednotlivé kapitoly jsou řazeny nejen podle složitosti, ale jsou především strukturovány tak, aby se zabývaly určitým tématem obsahující ukázkové příklady s popisem jejich tvorby.

Klíčová slova: TeX, LaTeX, Picture, PSTricks, PGF & Ti*k*Z, grafické ilustrace

Abstract

The goal of my bachelor thesis is to describe how to operate with package called TikZ using typesetting of pictures by system called LATEX. The whole work is written as a manual meant especially for beginners. It is devided into several chapters, each of them is focused on creation of pictures in computer science and also in electrical engineering. The chapters are ordered by dificulty and their structure shows particular topic as well as described examples are included.

Keywords: TeX, LaTeX, Picture, PSTricks, PGF & Ti*k*Z, graphic illustrations

Seznam použitých zkratek a symbolů

PGF	–	Portable Graphics format
Ti*k*Z	–	TikZ ist kein Zeichenprogramm
CSV	–	Comma Separated Value(s)
HTML	–	Hyper Text Markup Language
PDF	–	Portable Document Format
PS	–	Post Script
PSTricks	–	Post Script Tricks
pt	–	jednotka bod
cm	–	jednotka centimetr
mm	–	jednotka milimetr

Obsah

Seznam tabulek

Seznam obrázků

Úvod

LaTeX (čti Latech) je velice mocným nástrojem pro tvorbu elektronických dokumentů. Je postaven na typografickém formátovacím programu TeX, který vytvořil americký profesor Donald Erwin Knuth. TeX je volně šiřitelný systém určený pro počítačovou sazbu knih, matematických textů, popřípadě jiných technických dokumentací. Pomocí tohoto systému je možné vytvořit soubor, obsahující text dokumentu a do něj vkládat speciální příkazy, které nám zaručí, jak má tento text být formátován. Ačkoliv je TeX výkonným systémem, je pro běžné použití příliš složitý a proto byly nad ním vytvořeny nadstavby, které umožňují snadnější a přirozenější zápis sázeného textu.

Nejznámější volně šiřitelná nadstavba je již dříve zmíněný LaTeX, jehož autorem je americký počítačový vědec Leslie Lamport. Jeho hlavní myšlenkou bylo zpřístupnit jazyk běžným uživatelům, kterým připadala práce se systémem TeX příliš složitá. Upravil současný TeX a vytvořil množinu maker a šablon, které umožňují uživatelům sázet text podle předem připravených formátů. S postupem času se může tato nadstavba pyšnit vysoké popularity, což má za následek rozšíření systému o nepřeberném množství různých balíčků. Jedním z těchto doplňků, kterému se budu ve své bakalářské práci věnovat, je balíček Ti*k*Z.

Balíček Ti*k*Z, který rozšiřuje možnosti systému LaTeX nabízí velice širokou škálu funkcí pro práci s grafikou. O jeho vzniku se postaral německý univerzitní profesor Till Tantau. Tento nástroj slouží k vytvoření velice kvalitních grafických ilustrací na bázi vektorové grafiky, která je známá svým kvalitním bezeztrátovým vykreslením.

Cílem mojí bakalářské práce je vytvořit uživatelskou příručku pro tento balíček, která by měla pokrývat nezbytné funkce pro vytváření grafických ilustrací, od jednoduchých geometrických objektů, po složitější obrázky vykreslující blokové diagramy, elektrotechnická schémata a logické obvody. Práce bude taktéž popisovat tvorbu obrázků reprezentující grafy naměřených hodnot a jednoduchých matematických funkcí. Práce bude obsahovat tvorbu jednoduché animace a způsob vykreslování obrázků v prezentaci. V poslední řadě se práce věnuje tvorbě vlastních tvarů. Příručka by měla sloužit naprostým začátečníkům, kteří se s tímto balíčkem nikdy nesetkali. Práce se bude skládat z více kapitol, které budou řazeny podle složitosti a různorodosti.

Stručný obsah jednotlivých částí:

Kapitola 1 se zabývá základními pojmy a principy. Kapitola taktéž představuje přehled vhodných editorů pro platformu Windows a Linux.

Kapitola 2 popisuje práci s jednoduchými útvary, kde je podrobně vysvětlen způsob jejich tvorby, využití barev a dalších vlastností.

Kapitola 3 se zabývá pozicováním uzlů za pomocí metod Chains a Matrix.

Kapitola 4 je věnována tvorbě blokových diagramů.

Kapitola 5 se zabývá sazbou obrázků s elektronickými schématy a logickými obvody.

Kapitola 6 popisuje způsob uložení hodnot do externího souboru CSV a jejím následným zobrazením v grafu. Druhá část je věnována sazbě jednoduchých matematických funkcí.

Kapitola 7 se zabývá tvorbou prezentací a animací.

Kapitola 8 je věnována tvorbě vlastních tvarů, které jsou vytvářeny příkazy základní vrstvy PGF.

1 Základní pojmy a principy

Systém TEX, který byl původně navržen pouze pro sazbu matematických textů, představoval pro mnohé začínající uživatele způsob, jak publikovat svá díla. I přes jeho velkou flexibilitu byl natolik složitý, že uživatele často odrazoval, proto se někteří vývojáři rozhodli, že vytvoří soubory maker, které jim usnadní práci při tvorbě vlastních publikací.

Pro sazbu velice rozsáhlých publikací, patří mezi nejznámější systém LATEX, který v sobě nese sadu nejpoužívanějších šablon a maker. Ty umožní nejen usnadnit práci začínajícím uživatelům, ale jsou také navrženy, aby správně dodržovaly typografické zásady (pravidla korektního vzhledu dokumentu). Vznik této nadstavby způsobil rapidní nárůst počtu uživatelů využívající tento systém. To ovšem s sebou přineslo i větší požadavky na funkcionalitu. Jedním z hlavních problémů bylo použití obrázků v dokumentu.

Jelikož systém TEX nebyl navržen pro začlenění grafiky do textu, byly všechny obrázky konstruovány programovacím jazykem METAFONT[1]. Tímto způsobem jsme mohli do dokumentu vytvářet kvalitní grafiku. Bohužel to s sebou přineslo i nevýhody, kdy jsme obrázky nemohli zobrazovat barevně a výstupem byla vždy rastrovaná grafika. Pro tvorbu barevných obrázků ve vektorovém tvaru byl na základě METAFONTu vyvinut METAPOST, který umožňuje obrázky ukládat do souboru v PostSriptu[2].

S postupem času byla vytvořena sada maker, která umožnila vytvářet postskriptové obrázky v systému TEX a LATEX ve velmi vysoké kvalitě. Nazývá se `PSTricks`, který byl od svého uvedení pravidelně aktualizován. Na rozdíl od prostředí `picture` určený pro LATEX, který dokáže vytvářet jednoduchou vektorovou grafiku, je `PSTricks` obsahující sadu mnoha nástrojů schopen vytvářet daleko propracovanější obrázky. Velmi brzo si tak získal širokou základnu uživatelů, kteří tak mohli vytvářet obrázky ve svých publikací. Ovšem přišlo se na to, že dokumenty využívající jazyk PostSript (zkráceně PS) nebylo možné sdílet mezi uživateli různých platforem. Firma Adobe, která stála za vznikem PS vyvinula formát PDF umožňující sdílet dokumenty nezávislé na aplikaci, hardwaru či platformě. Tento formát se stal velice oblíbený pro elektronickou publikaci, tvorbu prezentací a jiných možností, které do něj byly postupně přidávány. To s sebou přineslo řadu problému pro uživatele `PSTricks`, neboť grafika vytvořená v tomto balíčku byla vždy překládána do PS. Z tohoto důvodu bylo navrženo spousta balíků, které zaručily konverzi do PDF formátu.

S jinou myšlenkou přišel profesor Till Tantau, který je autorem jednoho z nejlepších balíků pro tvorbu prezentací. Vytvořil balík PGF & Ti*k*Z sloužící pro tvorbu vektorové grafiky, který by umožňoval vytvářet soubory ve formátu PDF případně PS. Další výhodou je určitě kompatibilita se zmíněným balíkem `Beamer`, jenže se stal velmi oblíbeným pro snadnou tvorbu profesionálních prezentací. Ačkoliv je PGF & Ti*k*Z na samém počátku svého vývoje, mnozí jej pasují do role nástupce balíku `PSTricks`.

PGF (Portable Graphic Format) se skládá z několika vrstev. Nejnižší vrstva je systémová, která obsahuje velké množství systémových příkazů, které mají na starosti modifikovat příkaz `\special` z důvodů možného překladu obrázků nezávisle na konkrétním programu. Základní vrstva poskytuje sadu příkazů pro tvorbu složitější grafiky, které mohou být složeny se vzá-

[1] jazyk pro generování fontů

[2] programovací jazyk sloužící k tvorbě vektorové grafiky

jemně propojených celků. Poslední je uživatelská vrstva Ti*k*Z obsahující soubor příkazů, které svojí syntaxí (kombinace METAFONT a PS) zjednodušují příkazy základní vrstvy. Vykreslování obrázků za pomocí této vrstvy je pro mnohé začínající uživatele způsob, jak ve svých publikací velmi jednoduše vytvářet kvalitní a propracovanou grafiku.

Abychom mohli začít vytvářet obrázky v balíčku Ti*k*Z je nutné nainstalovat systém LaTeX a kvalitní editor pro efektivnější psaní zdrojových souborů. Musíme si ovšem rozmyslet, na které platformě chceme editor používat.

1.1 Windows

Mezi velice oblíbené editory pro platformu Windows patří TeXnicCenter[3] a Texmaker[4]. Nejen, že jsou oba zdarma, ale nabízí také velice přívětivé prostředí, které je navíc doplněno o velké množství funkcí. Druhý jmenovaný je dokonce k dispozici v češtině.

Dříve než se rozhodneme pro konkrétní editor, musíme si nainstalovat sázecí systém obsahující kompilátor LaTeXu. Pro Windows jsou zcela zdarma dostupné systémy TeXLive[5] a MiKTeX[6]. Oba navíc obsahují sadu podpůrných balíčků, včetně požadovaného balíčku Ti*k*Z.

Poslední věcí je instalace jednoduchého prohlížeče PDF. Podobně jako u editorů máme k dispozici velké množství nástrojů pro prohlížení souborů ve formátu PDF. Velice známe nástroje, které si můžeme pořídit zcela zdarma jsou Sumatra[7] a Acrobat Reader[8]. Pokud chceme rychlý a nenáročný prohlížeč použijeme Sumatru. Ta ovšem na úkor rychlosti a jednoduchosti nepodporuje animace. Zatímco druhý jmenovaný animace podporuje, avšak je z důvodu velkého množství podporovaných funkcí často zpomalován.

1.2 Linux

V případě, že chceme naše dokumenty vytvářet na platformě Linux, můžeme pro naši práci použít editor Texmaker, který byl vyvinut taktéž pro Linux a Mac. V jiném případě můžeme použít např. editor Kile. Pod Linuxem se nejčastěji používá sázecí systém TeXLive. Pro instalaci všech potřebných programů, použijeme následující příkazy:

Nainstaluje základní edici systému TeXLive.

```
sudo apt-get install texlive
```

Nainstaluje podporu češtiny a slovenštiny

```
sudo apt-get install texlive-lang-czechslovak
```

Instalace editoru Texmaker/Kile

```
sudo apt-get install texmaker
```

[3] http://www.texniccenter.org
[4] http://www.xm1math.net/texmaker
[5] http://www.tug.org/texlive
[6] http://www.miktex.org/
[7] http://blog.kowalczyk.info/software/sumatrapdf/download-free-pdf-viewer.html
[8] http://www.adobe.com/products/reader.html

```
sudo apt-get install kile
```

Pro tvorbu bakalářské práce jsem použil textový editor TeXnicCenter s kombinací se sázecím systémem MiKTEX. Podle potřeby jsem využíval oba nástroje k prohlížení PDF souborů.

2 Práce s jednoduchými objekty

Pokud se rozhodneme v dokumentu využívat grafiku, kterou nám nabízí balík Ti*k*Z, je nutné ho nejprve zavést. Pro zavedení balíčku provedeme jednoduchý příkaz:

```
\usepackage{tikz}
```

Tento příkaz vložíme do preambule, což je úsek, kde se nacházejí příkazy, jejichž platnost se vztahuje k celému dokumentu (např. nastavení rozměrů stránky, importování balíčku pro zavedení češtiny, atd.). Jakmile máme balíček uložený do preambule, můžeme začít vytvářet grafické ilustrace. Pokud se jedná o velice jednoduchý objekt, je možné jej kreslit do textu. Musíme ovšem před objektem zadefinovat příkaz `\tikz`, čímž dáme najevo, že se jedná o obrázek nikoliv o text. Druhá varianta je určena pro složitější ilustrace, které se musí vkládat do prostředí ohraničené dvojicí příkazů `\begin{tikzpicture}` a `\end{tikzpicture}`.

```
\documentclass[a4paper,12pt]    %definovaná třída
      %preambule
\userpackage{tikz}    %balík maker TikZ
\usepackage[czech]{babel}
\begin{document}    %začátek dokumentu
\begin{tikzpicture}
          %zde vložíme příkazy pro tvorbu grafiky
\end{tikzpicture}
\end{document}    %konec dokumentu
```

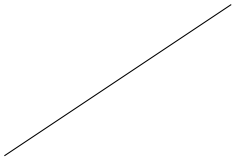
V této kapitole si na jednoduchých grafických ilustrací ukážeme práci s příkazy `draw`, `fill` a `node`. Stavebním kamenem všech obrázků obsažených v balíku Ti*k*Z je použití příkazu `path`. Pro vykreslení jednoduchého objektu slouží následující příkaz `\path[draw]`, kde se obvykle používá zkrácená forma `\draw`. Příkaz `fill` se zaměřuje na vyplnění ohraničených objektů barvou. Poslední příkaz `node` umožňuje přidávat popisky ke zvoleným bodům.

2.1 Přímky a křivky

Pro pochopení jakým způsobem můžeme vytvořit přímku nám poslouží tento vzor.

```
\draw (x0,y0) -- (x1,y1) -- ... -- (xn,yn);
```

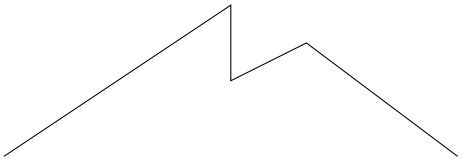
Samotné vykreslení se provádí pomocí souřadnicového systému. Tudíž pro vykreslení jednoduché přímky o dvou bodech je třeba nadefinovat počáteční bod (x0,y0) a koncový bod (x1,y1).



Obrázek 2.1: Vykreslení přímky

```
\draw (-1,-1) -- (2,1);
```

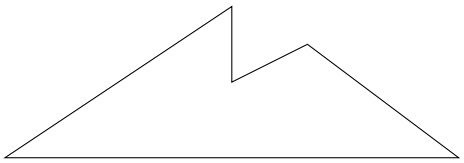
V dalším příkladu je vyobrazena křivka, začínající bodem (0,0) až do koncového bodu (6,0), která prochází přes všechny zadané body.



Obrázek 2.2: Vykreslení křivky

```
\draw (0,0) -- (3,2) -- (3,1) -- (4,1.5) -- (6,0);
```

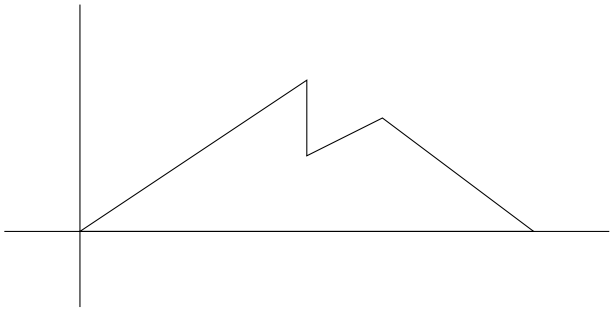
Může nastat situace, kdy je nutné spojit počáteční bod s koncovým. Provede se to zadáním klíčového slova `cycle` na konec příkazu.



Obrázek 2.3: Vykreslení spojitého objektu

```
\draw (0,0) -- (3,2) -- (3,1) -- (4,1.5) -- (6,0) -- cycle;
```

Ovšem samotné vykreslení přímky popřípadě křivky nám toho moc neřekne. Proto do obrázku s křivkou (Obrázek 2.2) vykreslíme další objekty, které budou reprezentovat dvojici os x a y. Abychom zajistili společné vykreslení všech objektů, umístíme uvedené příkazy do ohraničeného prostředí `tikzpicture`.

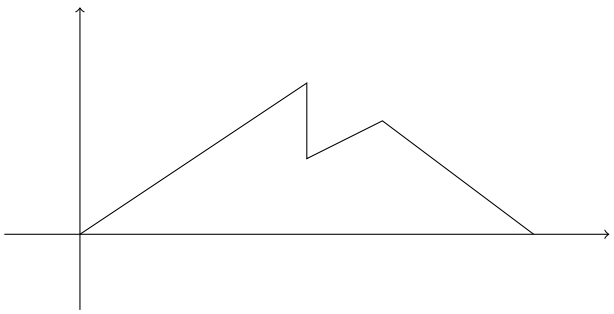


Obrázek 2.4: Společné vykreslení křivky, osy x a y

```
\begin{tikzpicture}
  \draw (-1,0)--(7,0); %vykresleni osy x
  \draw (0,-1)--(0,3); %vykresleni osy y
  \draw (0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
\end{tikzpicture}
```

2.2 Popisky, šipky a jiné zakončení

V předchozím obrázku jsme si ukázali, jak lze jednoduše vytvořit dvojici os. Je nutné dodržovat jistá pravidla geometrie, proto upravíme předešlý obrázek tak, že objekty představující osy, zakončíme jednosměrnými šipkami. Za příkaz `draw` vložíme nepovinný parametr `->`, který je umístěn do bloku uvozený hranatými závorkami.

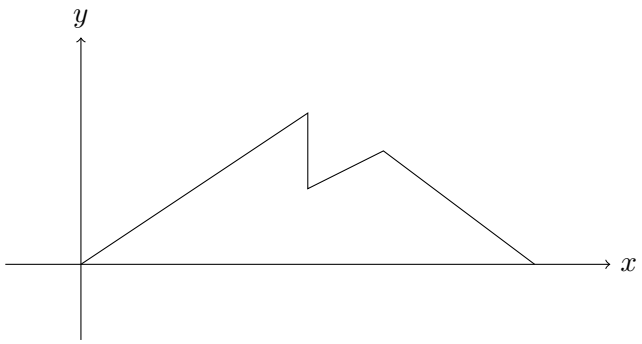


Obrázek 2.5: Zakončení objektů jednosměrnými šipkami

```
\begin{tikzpicture}
  \draw [->](-1,0)--(7,0);
  \draw [->](0,-1)--(0,3);
  \draw (0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
\end{tikzpicture}
```

Existují i jiné možnosti zakončení:
`[->>]` ↠ `[|-]` ⟝ `[->|]` ↛ `[latex-]` ← `[-to]` → `[stealth-]` ←
Můžeme použít i obousměrné zakončení, popřípadě lze šipky různě kombinovat.
`[<->]` ↔ `[<<->>]` ↞↠ `[|-latex]` ⟝▸ `[|<-to|]` ↤↦

Použitím příkazu `node` nám umožní ke každému určenému bodu přiřadit popisek. Použijeme tedy náš současný graf, ve kterém si obě osy pojmenujeme. Za klíčovým slovem `node` definujeme v hranatých závorkách parametr, který bude představovat umístění popisku kolem zvoleného bodu. Do složených závorek mezi dvojici znaků `$$` vložíme text, který chceme u bodu zobrazovat.



Obrázek 2.6: Přiřazení popisků k jednotlivým osám

```
\begin{tikzpicture}
  \draw [->](-1,0)--(7,0) node [right] {$x$}; %prirazeni popisku k bodu (7,0)
  \draw [->](0,-1)--(0,3) node [above] {$y$}; %prirazeni popisku k bodu (0,4)
  \draw (0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
\end{tikzpicture}
```

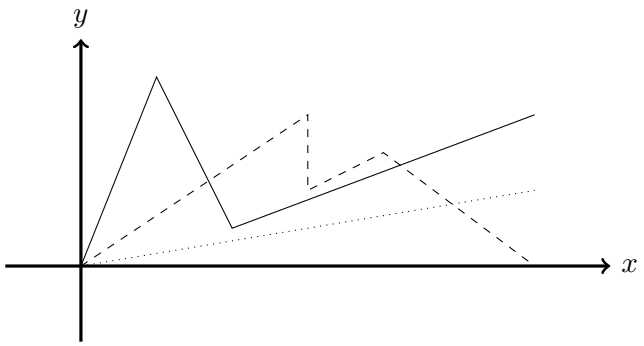
2.3 Změna tloušťky a typu čáry

Pro každý objekt můžeme v parametru libovolně nastavit různé zobrazení čáry.

Typ čáry	Argument	Popis
	[thin]	normální čára
	[very thin]	tenčí čára
	[ultra thin]	nejtenčí čára
	[thick]	tlustá čára
	[very thick]	tlustší čára
	[ultra thick]	nejtlustší čára
	[dashed]	čárkovaná čára
	[loosely dashed]	čárkovaná čára s většími mezerami
	[densely dashed]	čárkovaná čára s menšími mezerami
	[dotted]	tečkovaná čára
	[loosely dotted]	tečkovaná čára s většími mezerami
	[densely dotted]	tečkovaná čára s menšími mezerami

Tabulka 2.1: Typy čar

V této části si ukážeme, jakým způsobem můžeme libovolně podle potřeby nastavit různé typy čar. Upravíme obrázek 2.6 tak, že k ose X a Y přiřadíme argument `very thick`, čímž docílíme tučného zobrazení daných objektů. Do obrázku následně vykreslíme další tři křivky, na které různě aplikujeme čáry z dané tabulky. Tímto krokem docílíme přehlednějšího zobrazení výsledného grafu.



Obrázek 2.7: Vykreslení křivek s odlišnými vzory čar

```
\begin{tikzpicture}
  \draw [dashed](0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
  \draw [dotted](0,0)--(6,1);
  \draw (0,0)--(1,2.5) -- (2,0.5)-- (6,2);
  \draw [->,very thick](-1,0)--(7,0) node [right] {$x$};
  \draw [->,very thick](0,-1)--(0,3) node [above] {$y$};
\end{tikzpicture}
```

Pro tloušťku čáry je možné využít již z přednastavených vzorů (viz. tabulka 2.1) nebo si můžeme nadefinovat vlastní. První řádek říká, jak široká bude čára v bodech (ve výchozím tvaru je nastavena na 1 pt), zatímco druhý řádek určuje šířku čáry v centimetrech, popř. v milimetrech.

Obrázek 2.8: Ukázka vlastní definice čáry

```
\draw [line width=3](0,0) -- (3,1);
\draw [line width=0.2cm](5,0) -- (2,1);
```

2.4 Použití barev

Jakýkoliv objekt můžeme jednoduše obtáhnout nebo plně vybarvit z následujících předem nad definovaných barev. Příkaz `fill` nám umožňuje vyplnit objekt barvou, zatímco příkaz `draw`, který se do hranatých závorek psát nemusí, pouze obtahuje objekty. Existuje kombinace obou příkazu `\filldraw`, který v daný moment objekty obtahuje i vyplňuje zároveň.

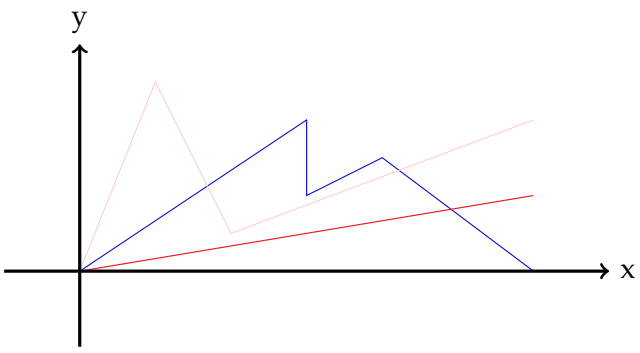
red green blue yellow
cyan magenta black gray
brown lime olive orange
pink purple teal violet
darkgray lightgray

Pokud by nám nestačily základní barvy, které máme k dispozici, je možné si namíchat barvu vlastní. Existují dva způsoby, jakým lze rozšířit repertoár barev. První možností je použít tzv. „vykřičníkovou“ metodu, kde za pomocí vykřičníku můžeme snížit sytost barvy popř. přimíchat barvu jinou. Druhá možnost je definovat si barvu vlastní s použitím příkazu:

```
\definecolor{název barvy}{rgb}
             {poměr červené, poměr zelené, poměr modré}
```

Poměr barev lze nastavit pouze v rozmezí 0-1.

Na obrázku 2.7 jsme mohli zpozorovat, že změna čáry nám velice usnadnila čtení v grafu. Takové znázornění je vhodné pouze v případě, že graf neobsahuje velké množství křivek. Daleko vhodnější způsob je označit každou křivku jinou barvou.



Obrázek 2.9: Vykreslení křivek s použitím barev

```
\begin{tikzpicture}
  \draw [blue](0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
  \draw [red](0,0)--(6,1);
  \draw [red!20](0,0)--(1,2.5) -- (2,0.5)-- (6,2);
  \draw [->,very thick](-1,0)--(7,0) node [right] {x};
  \draw [->,very thick](0,-1)--(0,3) node [above] {y};
\end{tikzpicture}
```

2.5 Geometrické útvary

Za pomocí křivek lze vytvořit jakýkoliv geometrický útvar od trojúhelníku, krychle, kvádru, kružnice a jiného tvaru. Jako příklad si vykreslíme obdélník o stranách 3 cm na 1 cm, který bude vyplněn červenou barvou.

Obrázek 2.10: Vykreslení obdélníku bod po bodu

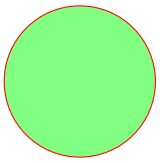
```
\draw [fill = red](0,0)--(3,0)--(3,1)--(0,1)--cycle;
```

Vykreslení takového obrazce bod po bodu je příliš složité a nepřehledné. Proto existují klíčová slova, které jasně definují o jaký objekt se má jednat. Totéž vykreslení pomocí klíčového slova `rectangle`. První závorka definuje bod začátku vykreslení, zatímco druhá závorka určuje délku stran (x,y).

Obrázek 2.11: Vykreslení obdélníku klíčovým slovem rectangle

```
\draw [fill = red!20](0,0) rectangle (3,1);
```

Příklad pro vykreslení kružnice, kde závorka (0,0) definuje bod středu a druhá určuje poloměr. V parametru definujeme za pomocí argumentů `draw` (obrys) a `fill` (výplň) odlišné zabarvení objektu.



Obrázek 2.12: Vykreslení kružnice

```
\draw [draw = red, fill = green!50](0,0) circle (1);
```

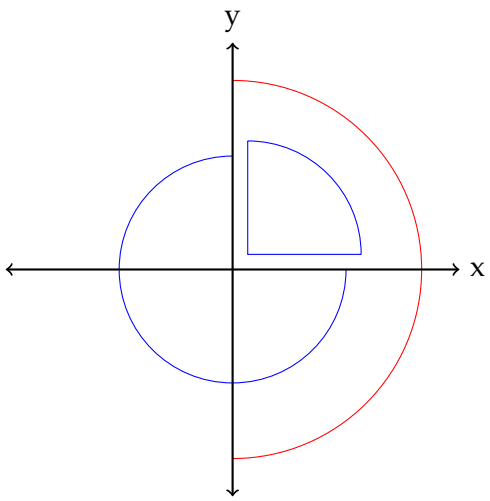
Příklad pro vykreslení elipsy, kde závorka (0,0) definuje bod středu a závorka (2 and 1) určuje šířku a výšku. Šířka je délka hlavní poloosy a výška je délka vedlejší poloosy. Objekt je vykreslen za pomocí příkazu `filldraw`, který zajistí stejnou barvu pro obrys a výplň.

Obrázek 2.13: Vykreslení elipsy

```
\filldraw[blue!20](0,0) ellipse (2 and 1);
```

Velice jednoduchým způsobem lze vykreslit kruhový oblouk. Abychom mohli vyobrazit pouze část kružnice, musíme nejdříve určit bod vykreslení (x,y) nebo-li také bod začátku od kterého se začne objekt vykreslovat. Následně za klíčovým slovem `arc` uvedeme tři komponenty umístěné v kulatých závorkách. Komponenty jsou od sebe odděleny dvojtečkami, kde první dvě složky jsou určeny pro úhly (α, β) a třetí definuje poloměr (r). Definováním klíčů `-|` a `|-` docílíme zalomením křivky do pravého úhlu.

`\draw (x,y) arc (`α`:`β`:r)`



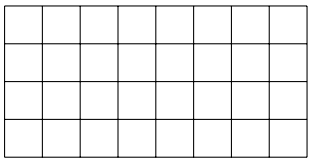
Obrázek 2.14: Vykreslení kruhového oblouku

```
\begin {tikzpicture}
  \draw [red](0,-2.5) arc (-90:90:2.5);
  \draw [blue](0,1.5) arc (90:360:1.5);
  \draw [blue](1.7,0.2) arc (0:90:1.5) (1.7,0.2) -| (0.2,1.7);

  \draw [<->,thick](-3,0)--(3,0) node [right] {x};
  \draw [<->,thick](0,-3)--(0,3) node [above] {y};
\end {tikzpicture}
```

2.6 Vykreslení mřížky, použití příkazu foreach

Pro vykreslení mřížky musíme nejdřív definovat počáteční bod, kterým řekneme odkud začne vykreslení. Poté za klíčovým slovem `grid` můžeme určit délku obou stran (x,y). V parametru můžeme pomocí argumentu `step` nadefinovat hustotu čar.

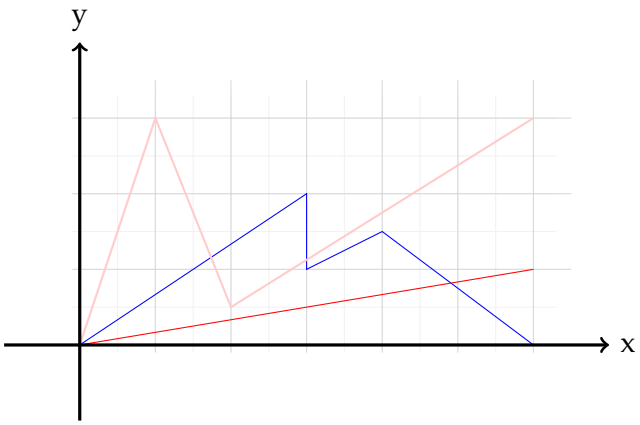
Tento příklad znázorňuje vykreslení mřížky od počátečního bodu (0,0) o délce 4 cm *(x)* na 2 cm *(y)*. Parametr **step** definovaný v hranatých závorkách nám zajistí vykreslení vodorovné a svislé čáry každých 0.5 cm.



Obrázek 2.15: Vykreslení mřížky o hustotě 0.5 cm

```
\draw [step=0.5](0,0) grid (4,2);
```

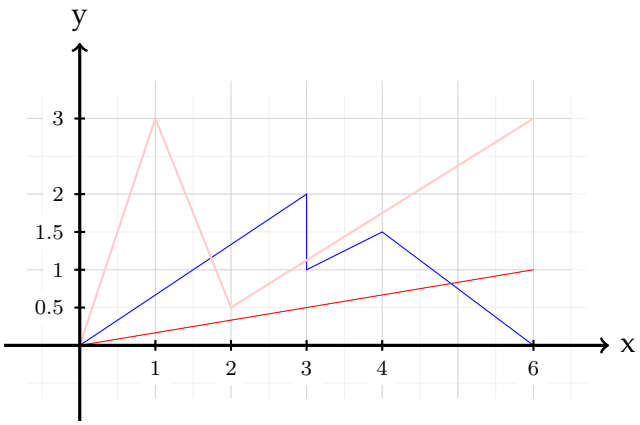
Do obrázku 2.9 nyní doplníme mřížku, která nám zajistí lepší orientaci v grafu.



Obrázek 2.16: Doplnění pomocných linek do grafu za pomocí mřížky

```
\begin{tikzpicture}
  \draw [gray!10, step=.5](-0.1,-0.1) grid (6.3,3.3);
  \draw [gray!30](-0.1,-0.1) grid (6.5,3.5);
  \draw [blue](0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
  \draw [red](0,0)--(6,1);
  \draw [red!20, thick](0,0)--(1,3)--(2,0.5)--(6,3);
  \draw [->,very thick](-1,0)--(7,0) node [right] {x};
  \draw [->,very thick](0,-1)--(0,4) node [above] {y};
\end{tikzpicture}
```

Obrázek můžeme rovněž rozšířit o body, které vyznačíme na ose X a Y. K tomuto kroku použijeme příkaz `foreach`. Tento příkaz nám umožní postupné procházení všech hodnot, které jsou uvedeny ve složených závorkách. Každá hodnota se postupně uloží do proměnné (v našem případě `\x`), která je dále využívána v příkazu `draw`. Jakmile cyklus `foreach` projde všechny hodnoty, které měl uloženy v závorce (pro osu X), ukončí se. Následně se provede i druhý příkaz `foreach` (pro osu Y).



Obrázek 2.17: Vyznačení a pojmenování bodů na osách x a y

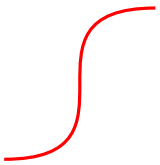
```
\begin{tikzpicture}
   %vykresleni mrizek
  \draw [gray!10, step=.5](-0.1,-0.1) grid (6.3,3.3);
  \draw [gray!30](-0.1,-0.1) grid (6.5,3.5);
   %vykresleni krivek
  \draw [blue](0,0)--(3,2)--(3,1)--(4,1.5)--(6,0);
  \draw [red](0,0)--(6,1);
  \draw [red!20, thick](0,0)--(1,3) -- (2,0.5)-- (6,3);
   %vykresleni os x a y
  \draw [->,very thick](-1,0)--(7,0) node [right] {x};
  \draw [->,very thick](0,-1)--(0,4) node [above] {y};
   %cyklus urceny pro vykreleni bodu na ose x
  \foreach \x in {1, 2 ,3, 4, 6}
  {
   \draw (\x cm,2pt)--(\x cm,-2pt)
      node[anchor=north, font=\scriptsize] {$\x$};
  }
   %cyklus urceny pro vykreleni bodu na ose y
  \foreach \y in {0.5, 1, 1.5, 2, 3}
  {
   \draw (2pt,\y cm)--(-2pt,\y cm)
      node[anchor=east, font=\scriptsize] {$\y$};
  }
\end{tikzpicture}
```

2.7 Béziérova křivka

Poslední příklad ilustruje jakým způsobem lze vytvořit Beziérovu křivku.

```
\draw (x0,y0) .. controls (x1,y1) and (x2,y2) .. (x3,y3);
```

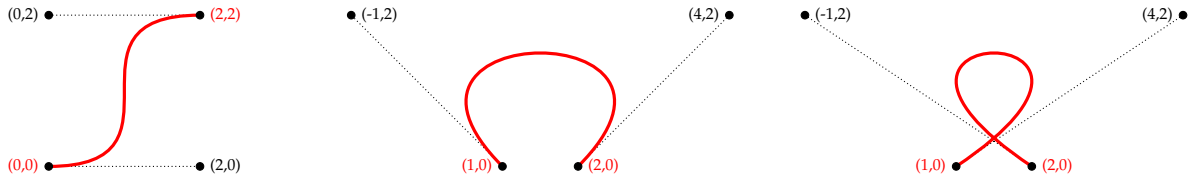
Beziérova křivka je tvořena pouze čtyřmi body, kde (x0,y0) je počátečním bodem a (x3,y3) je koncovým bodem křivky. Body (x1,y1) a (x2,y2) reprezentují dva středové kontrolní body, kterými určujeme tvar křivky.



Obrázek 2.18: Vykresleni Béziérovy křivky

```
\draw [red, very thick ](0,0) .. controls (2,0) and (0,2).. (2,2);
```

Pro lepší představivost, jakým způsobem můžeme měnit tvar křivek za pomocí dvou kontrolních bodů nám slouží následující obrázky.



Obrázek 2.19: Ukázky použití Béziérovy křivky

3 Pozicování uzlů - metody Chains a Matrix

V této kapitole si ukážeme, jak snadno můžeme napozicovat objekty. Pro jejich umístění na správně zvolená místa použijeme dvě metody: `chains` (řetězení) `matrix` (matice). Metoda řetězení nám umožňuje vytvořit sekvenci připojených objektů, zatímco matice slouží pro zarovnání objektů pomocí řádků a sloupců.

Abychom si podrobněji vysvětlili obě metody, je nutné si nejdříve připomenout práci s uzly (`node`).

3.1 Práce s příkazem node

V předchozí kapitole jsme příkaz `node` použili pouze pro přiřazení popisku ke zvolenému bodu. Je to velice mocný nástroj, jehož vlastnosti umístěné v hranatých závorkách blíže specifikují možnosti vykreslení. Uzel si taktéž můžeme pojmenovat, což nám umožní se na něj později odkázat.

Pro ukázku si vytvoříme uzel, který si pojmenujeme jako „tvarNode“. Na uzel se poté aplikují různé vlastnosti (tvar, barva obrysu, barva výplně, ...). Jestliže nezvolíme bod umístění, pak bude uzel ve výchozím stavu nastaven na pozici (0,0).

Obrázek 3.1: Vytvoření uzlu

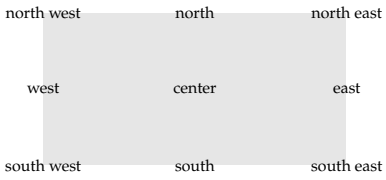
```
\begin{tikzpicture}
  \node (tvarNode)          %vytvoreni a pojmenovani uzlu
    [rectangle,             %tvar uzlu
    top color=white,        %vrchni barva vyplne
    bottom color=gray!50,   %postupny prechod do barvy uprostred
    draw=black!50,          %barva obtazeni
    very thick,             %tloustka obrysu
    minimum height=1cm,     %minimalni vyska tvaru
    minimum width=2cm,      %minimalni sirka tvaru
    rounded corners=3mm]    %zaobleni rohu tvaru
    {tvar};                 %popisek
\end{tikzpicture}
```

V případě, že bychom chtěli vytvořit více uzlů obsahující stejné vlastnosti, je použití této metody nevhodné. Z tohoto důvodu si v hranatých závorkách v prostředí `tikzpicture` nadefinujeme vlastní styl, který si pojmenujeme a ten poté použijeme u vytvořených uzlů. Druhý uzel umístíme napravo ve vzdálenosti 0.5 cm.

Obrázek 3.2: Použití vlastního stylu

```
\begin{tikzpicture}
   [mujStyl/.style={          %vytvoreni vlastniho stylu
    rectangle,
    top color=white,
    bottom color=gray!50,
    draw=black!50,
    very thick,
    minimum height=1cm,
    minimum width=2cm,
    rounded corners=3mm},
    node distance=.5cm]       %nastaveni vzdalenosti mezi uzly

    %vytvoreni uzlu s pouzitim vlastniho stylu
    %umisteni uzlu tvarNode2 doprava od uzlu tvarNode
    \node (tvarNode)[mujStyl] {tvar};
    \node (tvarNode2)[mujStyl,right=of tvarNode] {tvar2};
\end{tikzpicture}
```

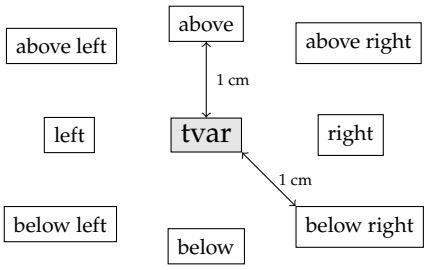
Ke každému vytvořenému uzlu můžeme připojovat další objekty (viz. Obrázek 3.2). Je však nutné uzel pojmenovat, čímž zajistíme odkaz na jeho pozdější volání a bližší specifikaci připojení k uzlu. Objekty se připojují k okraji popř. na střed každého uzlu. Pro ukázku připojovacích bodů slouží následující obrázek.



Obrázek 3.3: Ukázka připojovacích bodů

```
\begin{tikzpicture}[popisky/.style={font=\tiny}]
 \node (tvarNode)
   [minimum height=2cm, minimum width=4cm, fill=gray!20] {};

 \node [popisky]at(tvarNode.west){west};
 \node [popisky]at(tvarNode.east){east};
 \node [popisky]at(tvarNode.north){north};
 \node [popisky]at(tvarNode.south){south};
 \node [popisky]at(tvarNode.center){center};
 \node [popisky]at(tvarNode.north west){north west};
 \node [popisky]at(tvarNode.north east){north east};
 \node [popisky]at(tvarNode.south west){south west};
 \node [popisky]at(tvarNode.south east){south east};
\end{tikzpicture}
```

Může nastat situace, kdy potřebujeme k danému uzlu umístit jiný uzel (např. přidat popisek). V tomto okamžiku musíme blíže určit umístění kolem zvoleného uzlu a vzdálenost mezi uzly. Pro ukázku slouží následující obrázek, kde jsme vytvořili uzel a ten pak obtékali dalšími uzly ze vše stran ve vzdálenosti 1 cm. Šipky následně propojují dva uzly mezi sebou.



Obrázek 3.4: Možnosti umístění kolem uzlu

```
\begin{tikzpicture}
  [mujStyl/.style={rectangle, draw, font=\scriptsize, node distance=1cm}]

  \node (tvarNode) [mujStyl, fill=gray!20, font=\normalsize] {tvar};

  \node [mujStyl, right=of tvarNode] {right};
  \node [mujStyl, above right=of tvarNode] {above right};
  \node [mujStyl, left=of tvarNode] {left};
  \node [mujStyl, above left=of tvarNode] {above left};
  \node [mujStyl, below=of tvarNode] {below};
  \node (belowRight)[mujStyl, below right=of tvarNode] {below right};
  \node (above)[mujStyl, above=of tvarNode] {above};
  \node [mujStyl, below left=of tvarNode] {below left};

  \draw [<->] (tvarNode.north) to node[font=\tiny, right]{1 cm} (above.south);
  \draw [<->] (tvarNode.south east) to node[font=\tiny, right]{1 cm} (belowRight.north west);
\end{tikzpicture}
```

3.2 Použití metody Chains

Metoda `chains` slouží pro vytvoření sekvence uzlů, které jsou obvykle uspořádány v řadě. Pojmenované uzly můžeme velice snadno propojit mezi sebou. Pokud chceme využívat kód určený pro řetězení objektů, je nutné do preambule zavést knihovnu `{chains}`.

```
\usetikzlibrary{chains}
```

Následující příklad ilustruje jednoduché řetězení uzlů. Abychom mohli na všechny uzly umístěné v prostředí `tikzpicture` aplikovat tuto metodu, je nutné rozšířit vlastnosti tohoto prostředí o argument `start chain`, který definuje začátek řetězce. Od tohoto okamžiku můžeme k tomuto řetězci připojovat uzly. Jestliže chceme aby byl uzel součástí řetězce, pak zavedeme do hranatých závorek argument `on chain`. Jelikož jsme nedefinovali vzdálenost a směr, jsou ve výchozím stavu uzly umístěny doprava a jsou od sebe vzdáleny 1 cm.

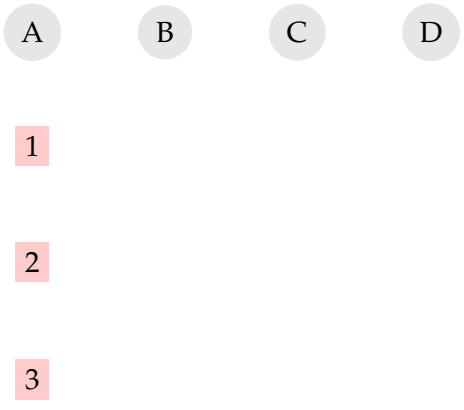
Obrázek 3.5: Zřetězení uzlů

```
\begin{tikzpicture}
   [start chain,
   mujStyl/.style={circle, fill=gray!20}]

   \node [on chain, mujStyl] {A};
   \node [on chain, mujStyl] {B};
   \node [on chain, mujStyl] {C};
   \node [on chain, mujStyl] {D};
\end{tikzpicture}
```

Argument `start chain` ovšem nemusíme pouze definovat v prostředí `tikzpicture`. Je možné vytvořit samostatný blok oddělený složenými závorkami, do kterého umístíme všechny uzly, které chceme zřetězit.

```
\begin{tikzpicture}
   [mujStyl/.style={circle, fill=gray!20}]

   { [start chain]
     \node [on chain, mujStyl] {A};
     \node [on chain, mujStyl] {B};
     \node [on chain, mujStyl] {C};
     \node [on chain, mujStyl] {D};
   }
\end{tikzpicture}
```

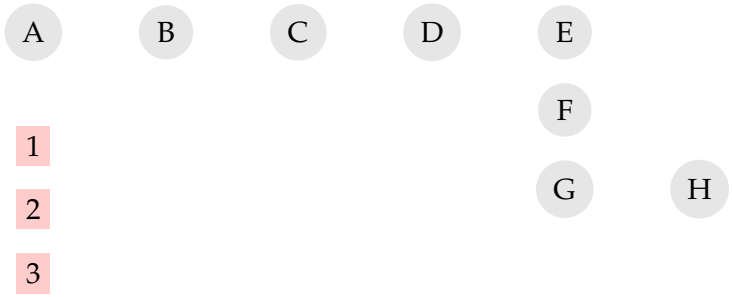
Tato knihovna nám umožňuje vytvořit více řetězení současně. Ovšem je nezbytné si jednotlivé řetězce pojmenovat. Další příklad zobrazuje vykreslení dvou řetězců, které si pojmenujeme jako „1“ a „2“. Oba řetězce jsme si nadefinovaly ve vlastnostech prostředí. První řetězec se vykreslí vodorovně, zatímco druhý svisle (začátek vykreslení řetězce „2“ určuje bod (0,-1.5)).



Obrázek 3.6: Použití více řetězců současně

```
\begin{tikzpicture}
   [ start chain=1 going right,
     start chain=2 going below,
     circleShape/.style={circle,fill=gray!20},
     rectangleShape/.style={rectangle,fill=red!20}]

     \node [on chain=1, circleShape] {A};
     \node [on chain=1, circleShape] {B};
     \node [on chain=1, circleShape] {C};
     \node [on chain=1, circleShape] {D};

     \node [on chain=2, rectangleShape] at (0,-1.5) {1};
     \node [on chain=2, rectangleShape] {2};
     \node [on chain=2, rectangleShape] {3};
\end{tikzpicture}
```

Další příklad vyobrazuje situaci, kdy k vytvořenému řetězci připojíme další uzly. Argument `continue chain` nám umožňuje pozměnit směr připojeného uzlu. Ve vlastnostech prostředí definujeme argument `node distance` pro odsazení všech uzlů vertikálně (0,3 cm) a horizontálně (1 cm).



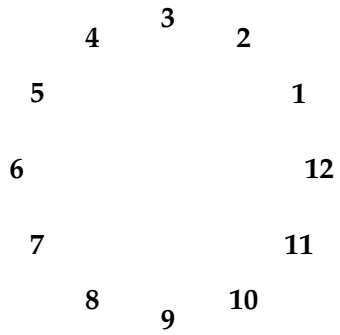
Obrázek 3.7: Změna směru řetězce

```
\begin{tikzpicture}
   [start chain=1 going right,
    start chain=2 going below,
    circleShape/.style={circle,fill=gray!20},
    rectangleShape/.style={rectangle,fill=red!20},
    node distance = 0.3cm and 1cm]

     \node [on chain=1, circleShape] {A};
     \node [on chain=1, circleShape] {B};
     \node [on chain=1, circleShape] {C};
     \node [on chain=1, circleShape] {D};

     \node [on chain=2, rectangleShape] at (0,-1.5) {1};
     \node [on chain=2, rectangleShape] {2};
     \node [on chain=2, rectangleShape] {3};

     \node [on chain=1, circleShape] {E};
     \node [on chain=1, continue chain=going below, circleShape] {F};
     \node [on chain=1, continue chain=going below, circleShape] {G};
     \node [on chain=1, circleShape] {H};
\end{tikzpicture}
```

Jako další příklad si vytvoříme jednoduché hodiny, na kterých si názorně předvedeme vykreslení číslic za pomocí kružnice. Nejdříve si nadefinujeme vlastnosti, které se budou aplikovat na všechny objekty uvnitř prostředí. Do hranatých závorek zahrneme argument `chain`, kterému přiřadíme tvar `circle` (kružnice). Na tento tvar se poté aplikuje makro `tikzchaincount`.

`tikzchaincount` `*` *úhel* : *poloměr kružnice*

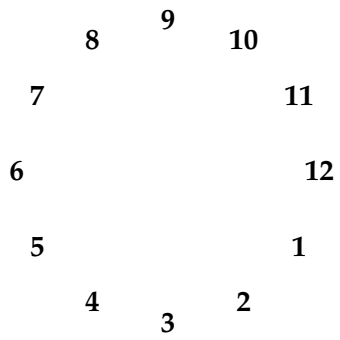
Toto makro slouží jako čítač, který se bude v průběhu procházení cyklu `foreach` zvyšovat o jedničku. Ten se následně vynásobí úhlem, který získáme tak, že podělíme obvod kružnice (360) počtem hodnot v cyklu (12), které budou reprezentovat hodiny na ciferníku. Výsledkem tohoto vzorce bude úhel o který se každá vykreslená hodnota na kružnici posune. Druhý parametr určuje poloměr kružnice. Cyklus bude postupně procházet všechny tyto hodnoty, které za pomocí příkazu `node` umístí na kružnici.



Obrázek 3.8: Vykreslení hodnot po celém obvodu kružnice

```
\begin{tikzpicture}[start chain=circle placed
      {at=(\tikzchaincount*30:2)}]
  \foreach \i in {1,...,12}
  \node [on chain] {\textbf{\i}};
\end{tikzpicture}
```

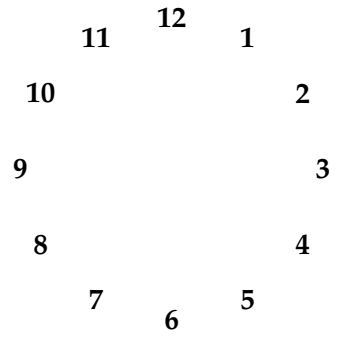
Tento krok nám umožnil, vysázet čísla po celém obvodu kružnice. Mezi každou číslicí je rozestup 30°. Ovšem vykreslení se provedlo proti směru hodinových ručiček. Tento problém můžeme vyřešit tak, že umístíme před úhel znaménko mínus, které nám zajistí, že se číslice budou vykreslovat opačným směrem.



Obrázek 3.9: Vykreslení hodnot ve směru hodinových ručiček

```
\begin{tikzpicture}[start chain=circle placed
      {at=(\tikzchaincount*-30:2)}] %pridani znamenka – pred uhel
  \foreach \i in {1,...,12}
  \node [on chain] {\textbf{\i}};
\end{tikzpicture}
```

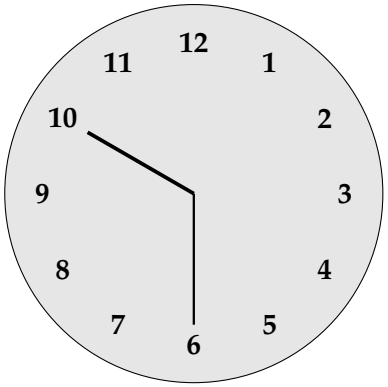
Ovšem stále to ještě není ono, neboť vykreslení se provádí od nulového úhlu. Vyřešit to můžeme tak, že otočíme kružnici o 90°, čímž docílíme korektního zobrazení hodin. Tato úprava se provede s použitím argumentu `rotate` uvnitř hranatých závorek v prostředí `tikzpicture`.



Obrázek 3.10: Otočení kružnice o 90°

```
\begin{tikzpicture}[start chain=circle placed
      {at=(\tikzchaincount*-30:2)}, rotate=90]
  \foreach \i in {1,...,12}
  \node [on chain] {\textbf{\i}};
\end{tikzpicture}
```

Abychom dosáhli požadovaného výsledku, je potřeba do obrázku ještě doplnit minutovou a hodinovou ručičku. Pro zvýraznění ciferníku vytvoříme kružnici, která bude sloužit jako pozadí.



Obrázek 3.11: Výsledné zobrazení ciferníku hodin

```
\begin{tikzpicture}[start chain=circle placed
      {at=(\tikzchaincount*-30:2)},rotate=90]
  \draw [fill=gray!20](0,0)circle(2.5); %pozadi ciferniku
  \foreach \i in {1,...,12}
  \node [on chain] {\textbf{\i}};
  \draw [line width = 0.7mm](0,0) -- (circle-10); %hodinova rucicka
  \draw (0,0) -- (circle-6); %minutova rucicka
\end{tikzpicture}
```

3.3 Použití metody Matrix

Matici můžeme chápat jako tabulku o *m* řádcích a *n* sloupcích. Do každé buňky této tabulky, můžeme vložit jakýkoliv objekt či popisek. Jestliže chceme objekty ukládat do matice, je zapotřebí nejprve zavést knihovnu `{matrix}`.

```
\usetikzlibrary{matrix}
```

Obrázek 3.12 vyobrazuje jednoduché napozicování uzlů pomocí matice. Skládá se ze třech sloupců a třech řádků. Každý řádek (včetně posledního) musí být ukončen znakem `\\`. Znak `&` slouží k oddělení buněk (sloupců). Všechny uzly, které chceme umístit do matice se musí nacházet v bloku `\matrix`.

1 2 3
4 5 6
7 8 9

Obrázek 3.12: Uzly umístěny do matice

```
\begin{tikzpicture}
 \matrix
   [fill=gray!10] %vypln ramecku matice
 {
   \node{1}; & \node{2}; & \node{3}; \\
   \node{4}; & \node{5}; & \node{6}; \\
   \node{7}; & \node{8}; & \node{9}; \\
 };
\end{tikzpicture}
```

Následující obrázek vykresluje tutéž matici jako v předchozím obrázku, ovšem s tím rozdílem, že jsme některé buňky nechali prázdné.

$$\begin{matrix} 1 & & 3 \\ & 5 & \\ 7 & & 9 \end{matrix}$$

Obrázek 3.13: Vykreslení matice s prázdnými buňkami

```
\begin{tikzpicture}
 \matrix
   [fill=gray!10]
 {
   \node{1};  &         &  \node{3}; \\
              &  \node{5};  &         \\
   \node{7};  &         &  \node{9}; \\
 };
\end{tikzpicture}
```

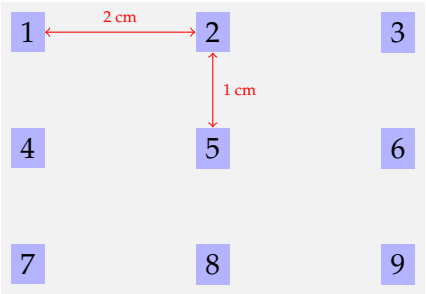
Je možné ve vlastnostech matice definovat argument `execute at empty cell`, který nám zajistí, že se do prázdných buněk matice doplní jiný objekt.

$$\begin{matrix} 1 & - & 3 \\ - & 5 & - \\ 7 & - & 9 \end{matrix}$$

Obrázek 3.14: Doplnění znaku - do prázdných buněk

```
\begin{tikzpicture}
 \matrix
   [fill=gray!10,
    execute at empty cell=\node{--};] %vsechny prazdne bunky,
                                      %se vyplni znakem —
 {
   \node{1};  &         &  \node{3}; \\
              &  \node{5};  &         \\
   \node{7};  &         &  \node{9}; \\
 };
\end{tikzpicture}
```

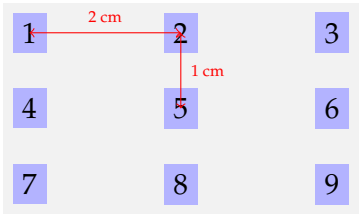
Na dalším obrázku si ukážeme, jak můžeme jednotlivé uzly od sebe odsadit. Ve vlastnostech bloku blíže specifikujeme odsazení řádků `row sep` a sloupců `column sep` od okraje každého uzlu.



Obrázek 3.15: Odsazení uzlů - od okraje

```
\begin{tikzpicture}
  \matrix
   [fill=gray!10,
    row sep=1cm,              %odsazeni radku od  okraje (ramecku) uzlu
    column sep=2cm,           %odsazeni sloupcu od okraje(ramecku) uzlu
    nodes={fill=blue!30}]     %vypln ramecku vsech uzlu
  {
   \node(n1){1};  &  \node(n2){2};  &  \node{3}; \\
   \node  {4};  &  \node(n5){5};  &  \node{6}; \\
   \node  {7};  &  \node  {8};  &  \node{9}; \\
  };

  %vykresleni  sipek
  \draw [<->,red] (n1.east) to node [font=\tiny, above]{2 cm} (n2.west);
  \draw [<->,red] (n2.south) to node[font=\tiny, right]{1 cm} (n5.north);
\end{tikzpicture}
```

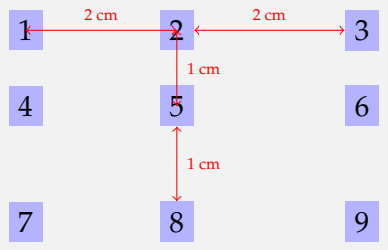
V obrázku 3.16 definujeme ve vlastnostech bloku argument `between origins`, který nám umožní odsadit řádky a sloupce od středu každého uzlu.



Obrázek 3.16: Odsazení uzlů - od středu

```
\begin{tikzpicture}
   \matrix
    [fill=gray!10,
     row sep={1cm,between origins},     %odsazeni radku od stredu uzlu
     column sep={2cm,between origins}, %odsazeni sloupcu od stredu uzlu
     nodes={fill=blue!30}]
   {
    \node(n1){1};  &  \node(n2){2}; &  \node{3}; \\
    \node  {4};  &  \node(n5){5};  &  \node{6}; \\
    \node  {7};  &  \node  {8};  &  \node{9}; \\
   };
   \draw [<->,red] (n1.center) to node [font=\tiny, above]{2 cm} (n2.center);
   \draw [<->,red] (n2.center) to node [font=\tiny, right]{1 cm} (n5.center);
\end{tikzpicture}
```

Další příklad zobrazuje situaci, kdy se rozhodneme, že pro jeden řádek/sloupec provedeme odsazení od středu uzlu a poté pro jiný řádek/sloupec uzly odsadíme od okraje uzlu. Argument `between borders` slouží pro odsazení řádků/sloupců od okraje uzlu.



Obrázek 3.17: Odsazení uzlů - od středu/od okraje

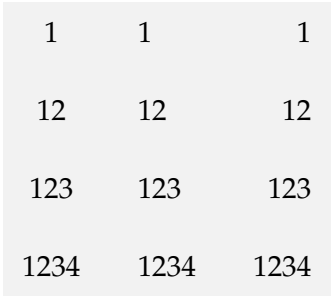
```
\begin{tikzpicture}
 \matrix
  [fill=gray!10,
   row sep={1cm,between origins},
   column sep={2cm,between origins},
   nodes={fill=blue!30}]
 {
  \node(n1){1};  &  \node(n2){2}; &  [between borders]\node(n3){3}; \\
  \node  {4};  &  \node(n5){5};  &  \node{6}; \\[between borders]
  \node  {7};  &  \node(n8){8};  &  \node{9}; \\
 };
 \draw [<->,red] (n1.center) to node [font=\tiny, above]{2 cm} (n2.center);
 \draw [<->,red] (n2.center) to node [font=\tiny, right]{1 cm} (n5.center);

 \draw [<->,red] (n2.east) to node [font=\tiny, above]{2 cm} (n3.west);
 \draw [<->,red] (n5.south) to node [font=\tiny, right]{1 cm} (n8.north);
\end{tikzpicture}
```

Na řádky, sloupce popř. samotné buňky můžeme aplikovat různé styly.

Obrázek 3.18: Použití stylu na řádek, sloupec a buňku

```
\begin{tikzpicture}
  \matrix
    [fill=gray!10,
     row sep=1mm,
     column sep=1mm,
     row 1/.style={nodes={fill=red}},                %pouziti stylu pro cely  radek  1
     column 1/.style={nodes={fill=green}},          %pouziti stylu pro cely sloupec 1
     row 3 column 3/.style={nodes={fill=yellow}}]   %pouziti stylu pro bunku (radek 3/sloupec 3)
   {
    \node{1};  &  \node{2};  &  \node{3}; \\
    \node{4};  &  \node{5};  &  \node{6}; \\
    \node{7};  &  \node{8};  &  \node{9}; \\
   };
\end{tikzpicture}
```

Objekty popř. popisky můžeme v matici zarovnat podle potřeby. Na následujícím příkladu jsme každý sloupec různě zarovnali. První sloupec je zarovnán na střed, druhý sloupec k levému okraji a třetí sloupec k pravému okraji.



Obrázek 3.19: Zarovnání sloupců

```
\begin{tikzpicture}
  \matrix
    [fill=gray!10,
     row sep=5mm, column sep=5mm,
     column 1/.style={anchor=base},      %zarovnani na stred
     column 2/.style={anchor=base west}, %zarovnani k levemu okraji
     column 3/.style={anchor=base east}] %zarovnani k pravemu okraji
   {
    \node{1};   &  \node{1};   &  \node{1};   \\
    \node{12};  &  \node{12};  &  \node{12};  \\
    \node{123};  &  \node{123};  &  \node{123};  \\
    \node{1234}; &  \node{1234}; &  \node{1234}; \\
   };
\end{tikzpicture}
```

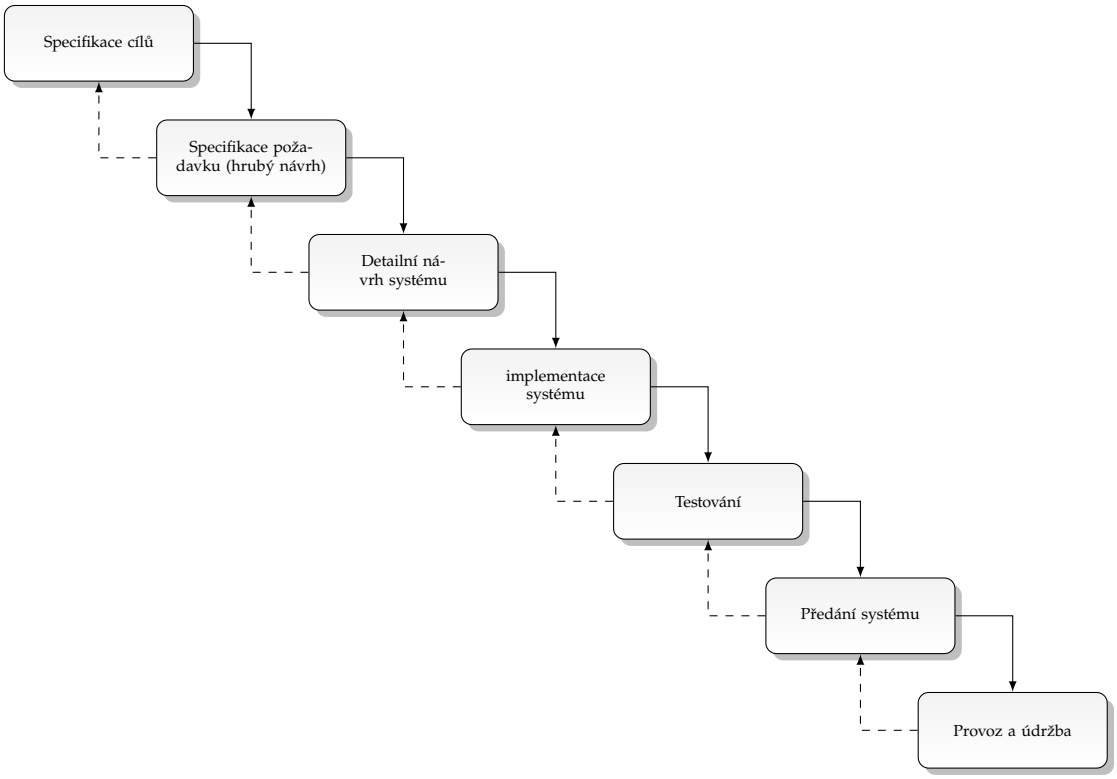
4 Bloková schémata

Obsahem této kapitoly je tvorba blokových schémat. Jak už z názvu vyplývá, důležitými články schématu jsou bloky a vztahy mezi nimi. Je zvykem bloky zobrazovat jako jednoduché geometrické obrazce (čtverec, obdélník, trojúhelník, kruh, atd.). Vztahy mezi těmito bloky jsou zpravidla znázorňovány šipkami, které udávají směr informačního toku z jednoho bloku do druhého.

V předchozí kapitole jsme si ukázali dvě metody pozicování. Podle složitosti a struktury diagramu se musíme rozhodnout, kterou metodu v daný okamžik použít.

4.1 Blokové schéma s použitím metody Chains

Následující obrázek znázorňuje blokový diagram, který popisuje životní cyklus softwaru (nazývá se *vodopádový model*). Tento model je složen z posloupnosti po sobě jdoucích bloků. Každý blok má svůj specifický význam a je vždy propojen se svým sousedem.



Obrázek 4.1: Blokový diagram I - vodopádový model

Abychom mohli vytvářet bloky, musíme si nejdříve nadefinovat vlastní styl, který nám zajistí, jak budou dané bloky vypadat. Tento styl pak aplikujeme na uzly, které budou reprezentovat samotné bloky. V předchozí kapitole jsme vlastní styly definovali uvnitř prostředí `tikzpicture`. Tato metoda je vhodná pouze za předpokladu, kdy styl nebudeme mimo prostředí využívat.

Problém ovšem nastává v případě, kdy potřebujeme ve více prostředí například vytvořit stejný obrazec. Jednou z možností je v každém prostředí definovat stejný styl. Toto řešení není příliš vhodné, proto existují globální styly. Globální styly můžeme volat ve více prostředí, neboť jsou viditelné v celém dokumentu. Příkaz pro vytvoření globálního stylu se nazývá `\tikzstyle`.

```
\tikzstyle {název stylu} [vlastnosti stylu]
```

Globální styl si pojmenujeme jako „block“, kde se za pomoci tohoto jména můžeme na tento styl odkazovat odkudkoliv v celém dokumentu. Ve vlastnostech (v hranatých závorkách) si pak nadefinujeme argumenty, které nám zaručí jak bude uzel vypadat a jak se bude chovat.

Na obrázku 4.2 je patrné, že jsme aplikovali argumenty, které nám vykreslí obdélník, jehož rohy jsou zaobleny. Je taktéž aplikována přechodová barevná výplň. Dále definujeme argumenty `text centered`, který nám zajistí, že bude popisek ve středu obdélníku, `text width`, který omezí délku popisku na 2 cm a `drop shadow`, který doplní stín za obdélník. Pro použití stínu je nutné v preambuli definovat knihovnu `{shadows}`.

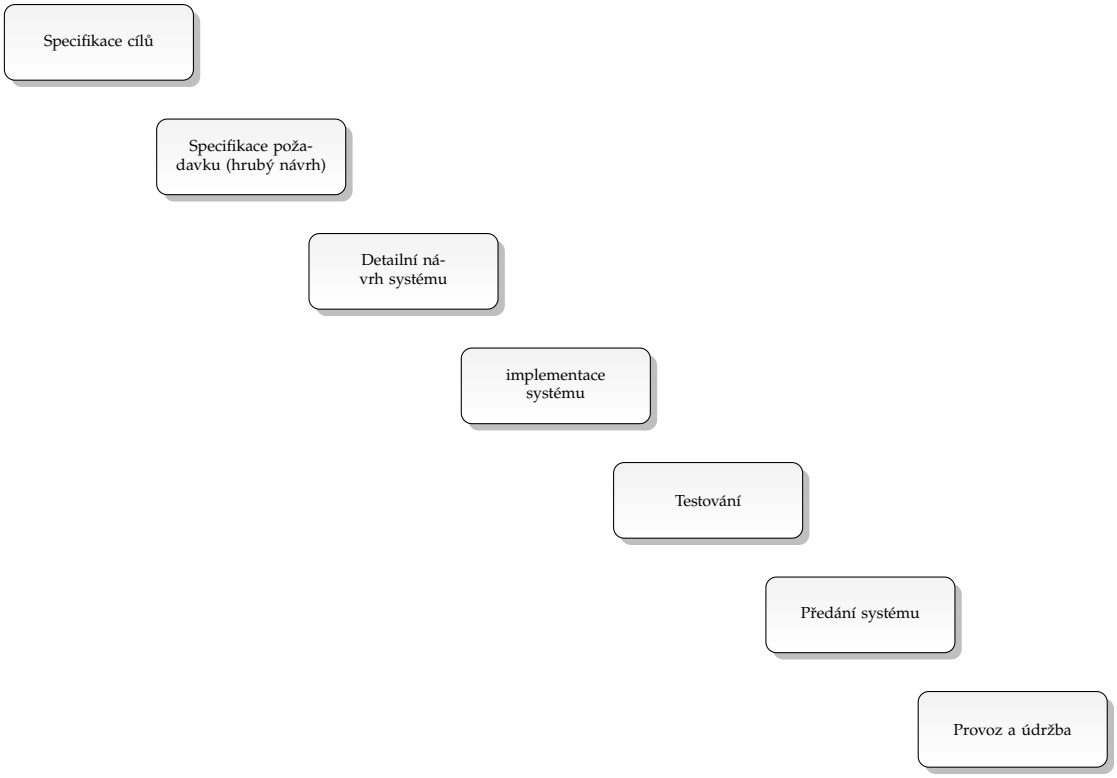
```
\usetikzlibrary {shadows}
```

Pro vykreslení obrazce použijeme uzel, u kterého definujeme globální styl. Uzel pro pozdější odkazování pojmenujeme jako „b1“.

Obrázek 4.2: Aplikování globálního stylu pro uzel

```
\tikzstyle{block}=[rectangle,draw,minimum height=1cm,
          minimum width = 2.5cm, text centered,text width=2cm,
          rounded corners, font=\tiny, top color=gray!10,
          bottom color=white,drop shadow]
\begin{tikzpicture}
   \node(b1)[block]{Specifikace cílů};
\end{tikzpicture}
```

Jakmile máme styl pro uzly vytvořený, musíme se rozhodnout jakým nejefektivnějším způsobem zobrazit ostatní uzly v této posloupnosti. Vhodným kandidátem pro pozicování se nabízí metoda `chains`, neboť můžeme uzly zobrazit jako řetězec. Ve vlastnostech prostředí si proto nadefinujeme argument `start chain`, kterému přiřadíme jméno a bližší specifikaci umístění uzlů. Jako další parametry zavedeme minimální výšku a minimální šířku obdélníku, velikost písma (popisku) a vzdálenosti mezi uzly od okraje (vertikálně and horizontálně). Na všechny uzly poté aplikujeme styl „block“ i metodu `chains`.



Obrázek 4.3: Zobrazení ostatních uzlů - použití metody chains

```
\tikzstyle{block}=[rectangle,draw,minimum height=1cm,
                minimum width = 2.5cm, text centered,text width=2cm,
                rounded corners, font=\tiny, top color=gray!10,
                bottom color=white,drop shadow]
\begin{tikzpicture}
  [start chain = 1 going below right,
   minimum height=1cm,
   minimum width =2.5cm,
   font=\tiny,
   node distance=.5cm and -.5cm]
    \node(b1)[on chain=1, block]{Specifikace cílů};
    \node(b2)[on chain=1, block]{Specifikace požadavku (hrubý návrh)};
    \node(b3)[on chain=1, block]{Detailní návrh systému};
    \node(b4)[on chain=1, block]{implementace systému};
    \node(b5)[on chain=1, block]{Testování};
    \node(b6)[on chain=1, block]{Předání systému};
    \node(b7)[on chain=1, block]{Provoz a údržba};
\end{tikzpicture}
```

Jako poslední věc, kterou je třeba provést jsou propojovací šipky (plné a přerušované). Pro vykreslení přechodu mezi uzly použijeme cyklus `foreach`. První cyklus vykreslí plné šipky od prvního obrazce k poslednímu, zatímco druhý cyklus začne vykreslovat přerušované šipky od posledního obrazce k prvnímu. Příkaz `pgfmathtruncatemacro` slouží pro definování

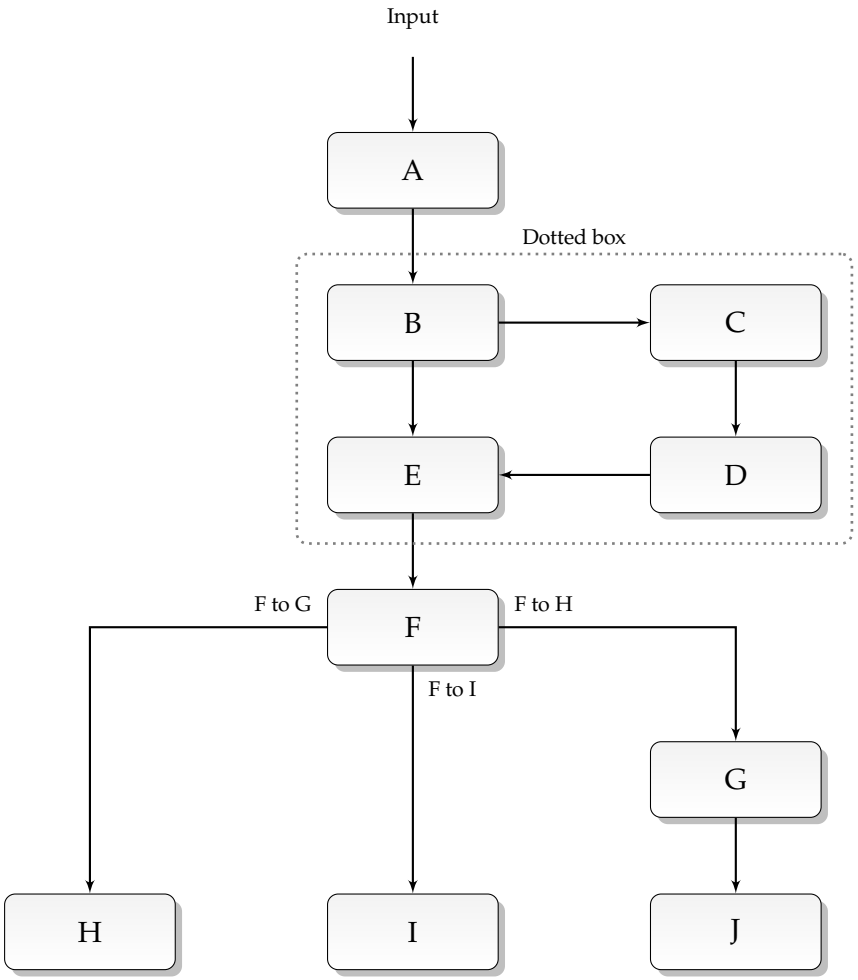
pomocné proměnné (tu si pojmenujeme), která je vždy zvýšena (u plné šípky) popř. snížena (u přerušované šipky) o hodnotu 1 (udává umístění druhého konce šipky).

Následující kód slouží pro vykreslení obrázku 4.1.

```
\tikzstyle{block}=[rectangle,draw,minimum height=1cm,
              minimum width = 2.5cm, text centered,text width=2cm,
              rounded corners, font=\tiny, top color=gray!10,
              bottom color=white,drop shadow]
\begin{tikzpicture}
  [start chain = 1 going below right,
   minimum height=1cm,
   minimum width =2.5cm,
   font=\tiny,
   node distance=.5cm and -.5cm]
    \node(b1)[on chain=1, block]{Specifikace cílů};
    \node(b2)[on chain=1, block]{Specifikace požadavku (hrubý návrh)};
    \node(b3)[on chain=1, block]{Detailní návrh systému};
    \node(b4)[on chain=1, block]{implementace systému};
    \node(b5)[on chain=1, block]{Testování};
    \node(b6)[on chain=1, block]{Předání systému};
    \node(b7)[on chain=1, block]{Provoz a údržba};

    \foreach \plna in {1,...,6} %vykresleni plne sipky
    {
      \pgfmathtruncatemacro\pomocnaPromenna{\plna+1}
      \draw[-latex](b\plna.east) -| (b\pomocnaPromenna.north);
    }
    \foreach \prerus in {7,...,2} %vykresleni prerusované sipky
    {
      \pgfmathtruncatemacro\pomocnaPromenna{\prerus-1}
      \draw[-latex,dashed](b\prerus.west) -| (b\pomocnaPromenna.south);
    }
\end{tikzpicture}
```

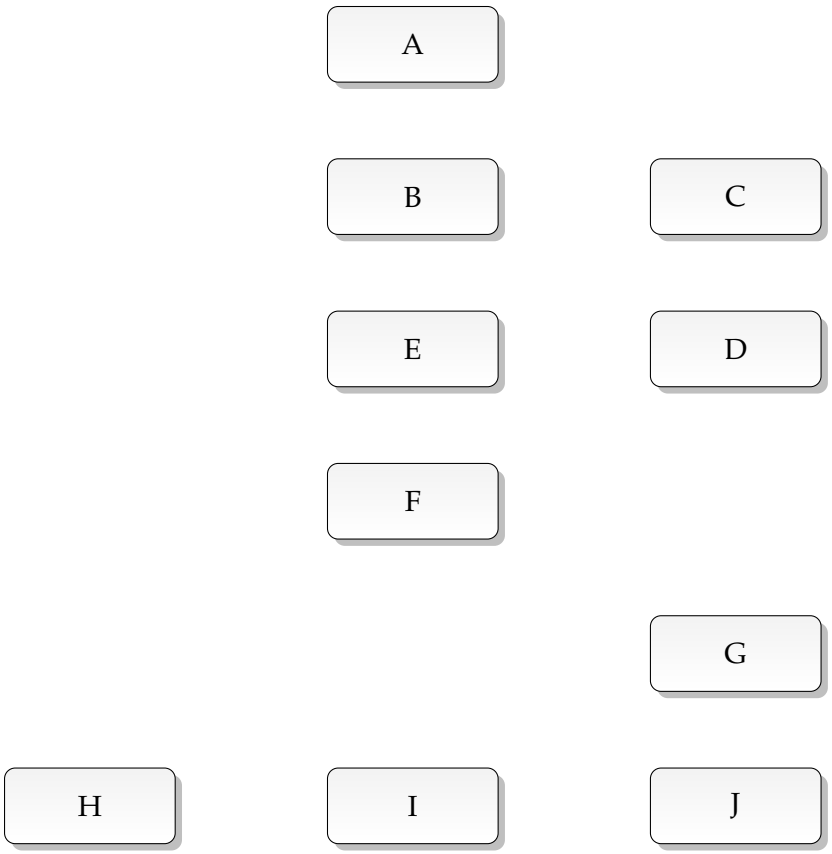
4.2 Blokové schéma s použitím metody Matrix

Blokové schéma na obrázku 4.4 je vykresleno pomocí metody `matrix`.



Obrázek 4.4: Blokový diagram II - Ukázkový příklad

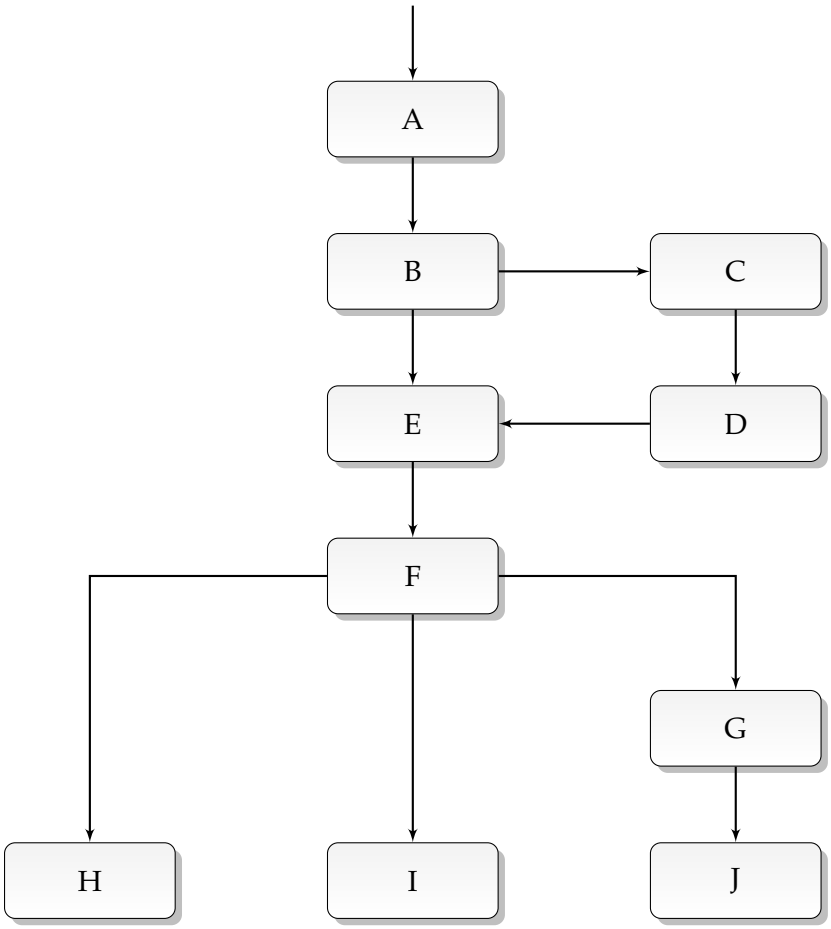
Z minulé kapitoly víme, že se vykreslení provádí za pomocí řádků a sloupců, tím nám vznikají buňky do kterých můžeme umístit jakýkoliv obrazec. Už na první pohled je patrné, že jsou bloky zarovnány a vykresleny podobně jako v tabulce. Každý vykreslený obrazec je umístěn do předem určené buňky. Je třeba si taktéž uvědomit, že bloky mají podobný vzhled jako v předchozím blokovém diagramu. Je to způsobeno použitím globálního stylu, definovaném v předchozím příkladu. Pro vykreslení obdélníků do matice nesmíme zapomenou přidat knihovnu `{matrix}` do preambule. Poté ve vlastnostech bloku `\matrix` uvedeme parametry, které budou určovat odsazení řádků/sloupců a minimální výšku/šířku obdélníku. Následně umístíme uzly do správných buněk.



Obrázek 4.5: Zobrazení ostatních uzlů - použití metody matrix

```
\begin{tikzpicture}
  \matrix
    [column sep=2cm, row sep=1cm,
     minimum height=1cm,
     minimum width=2cm]
  {
     & \node(p_start){};    &                   \\
     & \node(b1)[block]{A};   &                 \\
     & \node(b2)[block]{B};   &  \node(b3)[block]{C}; \\
     & \node(b5)[block]{E};   &  \node(b4)[block]{D}; \\
     & \node(b6)[block]{F};   &                 \\
     &                      &  \node(b7)[block]{G}; \\
     \node(b8) [block]{H};    &
     \node(b9) [block]{I};    &
     \node(b10)[block]{J};                      \\
  };
\end{tikzpicture}
```

Nyní si vykreslíme šipky, jejíž tvar si nadefinujeme v globálním stylu, který si nazveme jako `line`.



Obrázek 4.6: Doplnění šipek

Uvnitř bloku `tikzpicture` můžeme vytvořit specifické prostředí `scope`, do kterého vložíme pouze příkazy, určené pro vykreslení šipek. Toto prostředí slouží jako oblast, kde můžeme zahrnout jen ty příkazy, které mají něco společného. Ve vlastnostech si poté nadefinujeme styl, který se vztahuje pouze na příkazy typu `path`. Tento styl se pak odkazuje na globální styl `line`. Pokud používáme takto samotný příkaz, musíme v preambuli naimportovat knihovnu, kterou budeme pro danou akci využívat (v našem případě chceme zobrazit šipky, tudíž potřebujeme knihovnu `arrows`).

```
\usetikzlibrary {arrows}
```

```
\tikzstyle{line}=[draw, thick, -latex']

\begin{tikzpicture}
  \matrix
    [column sep=2cm,
     row sep=1cm,
     minimum height=1cm,
     minimum width=2cm]
  {
     & \node(p_start){};    &                   \\
     & \node(b1)[block]{A};   &                 \\
     & \node(b2)[block]{B};   &  \node(b3)[block]{C}; \\
     & \node(b5)[block]{E};   &  \node(b4)[block]{D}; \\
     & \node(b6)[block]{F};   &                 \\
     &                      &  \node(b7)[block]{G}; \\
     \node(b8) [block]{H};    &
     \node(b9) [block]{I};    &
     \node(b10)[block]{J};                      \\
  };

  %Oblast urcena pro vykresleni sipek
  \begin{scope} [every path/.style = line]
    \path (p_start) -- (b1);
    \path (b1)    -- (b2);
    \path (b2)    -- (b3);
    \path (b3)    -- (b4);
    \path (b4)    -- (b5);
    \path (b2)    -- (b5);
    \path (b5)    -- (b6);
    \path (b6)    -| (b7);
    \path (b7)    -- (b10);
    \path (b6)    -| (b8);
    \path (b6)    -- (b9);
  \end{scope}
\end{tikzpicture}
```

Může nastat situace, kdy potřebujeme v blokovém schématu zvýraznit popř. vymezit určitou oblast. Ve většině vývojových diagramů se taková akce provádí čárkovaným případně tečkovaným obrysem. Nejdříve si vytvoříme globální styl `gray dotted`, do kterého vypíšeme potřebné parametry. Argumentem `inner sep` provedeme odsazení od všech objektů, které jsou do této oblasti zahrnuty.

Poté do nového prostředí `scope` vložíme uzel, který bude reprezentovat tečkovanou oblast. Ve vlastnostech uzlu následně definujeme styl `gray dotted` který chceme aplikovat a argument `fit`, do kterého zahrneme všechny objekty, které chceme mít do této oblasti umístěny.

Pozor! Je důležité mít na paměti, že se oblast bude vykreslovat podle tvaru, který definujeme v globálním stylu.

Poslední `scope` bude obsahovat popisky u zvolených uzlů.

Následující kód slouží pro vykreslení obrazku 4.4.

```
\tikzstyle{line}=[draw, thick, -latex']

\tikzstyle{gray dotted}=[rectangle, dotted, draw=gray, line width=1pt,
     inner sep=4mm, rounded corners]

\begin{tikzpicture}
 \matrix
   [column sep=2cm,
    row sep=1cm,
    minimum height=1cm,
    minimum width=2cm]
 {
    & \node(p_start){};    &                  \\
    & \node(b1)[block]{A};   &                  \\
    & \node(b2)[block]{B};   &  \node(b3)[block]{C}; \\
    & \node(b5)[block]{E};   &  \node(b4)[block]{D}; \\
    & \node(b6)[block]{F};   &                  \\
    &                      &  \node(b7)[block]{G}; \\
    \node(b8) [block]{H};    &
    \node(b9) [block]{I};    &
    \node(b10)[block]{J};                       \\
 };

 %Oblast urcena pro vykresleni sipek
 \begin{scope} [every path/.style = line]
   \path (p_start) -- (b1);
   \path (b1)    -- (b2);
   \path (b2)    -- (b3);
   \path (b3)    -- (b4);
   \path (b4)    -- (b5);
   \path (b2)    -- (b5);
   \path (b5)    -- (b6);
   \path (b6)    -| (b7);
   \path (b7)    -- (b10);
   \path (b6)    -| (b8);
   \path (b6)    -- (b9);
 \end{scope}

 %Prostredi urcene teckovane oblasti
 \begin{scope}
   \scriptsize
   \node (box) [gray dotted, fit = (b2) (b3) (b4)] {};
   \node at (box.north) [above, inner sep=1mm] {Dotted box};
 \end{scope}

 %Prostredi urcene popiskum
 \begin{scope}
   \scriptsize
   \node at (p_start){Input};
   \node at (b6.east)[above right, inner sep=2mm]{F to H};
   \node at (b6.west)[above left, inner sep=2mm]{F to G};
   \node at (b6.south)[below right, inner sep=2mm]{F to I};
 \end{scope}
\end{tikzpicture}
```

5 Elektronická schémata a logické obvody

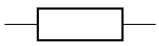
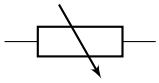
V této kapitole si na jednoduchých příkladech ukážeme sazbu obrázků znázorňující elektronická schémata a logické obvody. Pro jejich vytvoření můžeme použít balíčky, které byly navrženy různými vývojáři, aby nám usnadnili práci při tvorbě jednoduchých či složitějších obrázků. Obsahem takového balíčku je sada prvků (schématických značek), které můžeme libovolně využívat podle potřeby. Balíčky pro tvorbu elektronických schémat a logických obvodů existuje mnoho. Lišit se mohou nejen vzhledem ale i samotnou syntaxí. Je poté pouze na nás, co nám vyhovuje nejlépe a jak se daný obrázek podobá naší představě. Mezi velice povedené balíky patří `circuitikz` [5], který disponuje poměrně jednoduchou syntaxí. K balíčku je rovněž přiložen velice kvalitní a přehledný manuál.

```
\usepackage {circuitikz}
```

Jelikož veškeré prvky jsou vytvořeny z geometrických tvarů, které jsou postaveny na balíčku Ti*k*Z, je nutné tyto prvky vkládat do prostředí `tikzpicture`, popř. do prostředí `circuitikz`.

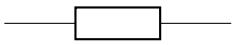
5.1 Elektronická schémata

Jako první si vykreslíme schématickou značku rezistoru. Pro bližší představu nám poslouží následující tabulka 5.1. Argumenty, které jsou popsány u každého prvku se definují v hranatých závorkách a představují danou schématickou značku. Některé značky lze definovat dvěma způsoby (vypsáním celého názvu, popř. zkrácenou formou). Musíme si uvědomit, že značky pro rezistor nejsou celosvětově sjednoceny. Evropa používá jiný symbol značení než ten, který se používá ve Spojených státech a Japonsku.

Značka	**Popis**	**Argument**
	rezistor US	[resistor] nebo [R]
	rezistor EU	[generic]
	rezistor US z proměnnou hodnotou	[variable resistor] nebo [vR]
	rezistor EU z proměnnou hodnotou	[tgeneric]

Tabulka 5.1: Schématické značky I

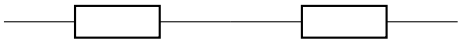
Na následujícím obrázku je vyobrazeno vykreslení rezistoru z jednoho bodu do druhého.



Obrázek 5.1: Vykreslení rezistoru

```
\begin{circuitikz}
  \draw (0,0) to[generic] (3,0);
\end{circuitikz}
```

Značky můžeme jednoduše vykreslit i za sebou (sériově). Oba kódy vykreslují stejný obrázek, liší se pouze zápisem (můžeme použít zkrácenou formu).



Obrázek 5.2: Vykreslení dvou rezistorů sériově

```
\begin{circuitikz}
  \draw (0,0) to[generic] (3,0);
  \draw (3,0) to[generic] (6,0);
\end{circuitikz}
```

Zkrácená forma zápisu.

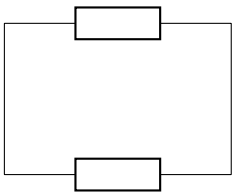
```
\begin{circuitikz}
   \draw (0,0) to[generic] (3,0) to[generic] (6,0);
\end{circuitikz}
```

Pro spojení dvou bodů se v `circuitikz` používá argument `short` (viz. tabulka 5.2). Je možné místo argumentu `short` použít pro spojení `--`.

Značka	Popis	Argument
	drát (čára)	[short] nebo --
	drát s uzlem na začátku a na konci	[short, *-*]
	drát se vstupem a výstupem	[short, o-o]

Tabulka 5.2: Schématické značky II

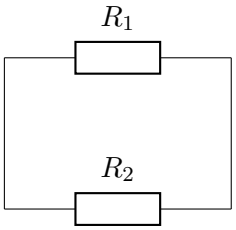
Obrázek 5.3 vyobrazuje vykreslení dvou rezistorů paralelně.



Obrázek 5.3: Vykreslení dvou rezistorů paralelně

```
\begin{circuitikz}
   \draw (0,0) to[generic] (3,0) to[short] (3,2) to[generic] (0,2) -- (0,0);
\end{circuitikz}
```

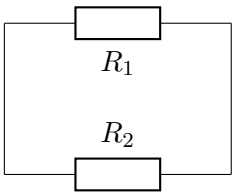
Pokud chceme ke značkám přidat popisky, uvedeme ve vlastnostech argument `l` (jako label) s textem, který chceme uvést.



Obrázek 5.4: Přidání popisků k rezistorům

```
\begin{circuitikz}
  \draw (0,0) to[generic, l=$R_2$] (3,0);
  \draw (0,2) to[generic, l=$R_1$] (3,2);
  \draw (0,0) -- (0,2);
  \draw (3,0) to[short] (3,2);
\end{circuitikz}
```

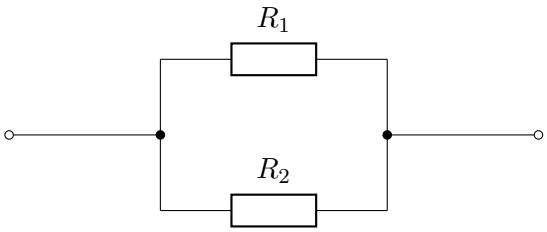
Pro určení pozice popisku kolem značky, můžeme použít znaky ^ pro obtékání ze shora/zleva, popř. _ pro obtékání ze zdola/zprava.



Obrázek 5.5: Změna umístění popisků

```
\begin{circuitikz}
   \draw (0,0) to[generic, l^=$R_2$] (3,0);
   \draw (0,2) to[generic, l_=$R_1$] (3,2);
   \draw (0,0) -- (0,2);
   \draw (3,0) to[short] (3,2);
\end{circuitikz}
```

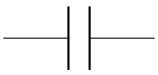
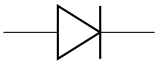
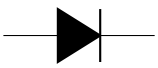
Rozšíříme obrázek 5.4 o vstup, výstup a uzly, které definují spojení.

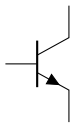
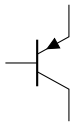


Obrázek 5.6: Doplnění vstupu, výstupu a spojovacích uzlů

```
\begin{circuitikz}
   %vykresleni rezistoru
   \draw (0,0) to[generic, l=$R_2$] (3,0);
   \draw (0,2) to[generic, l=$R_1$] (3,2);
   %spojovaci cary
   \draw (0,0) -- (0,2);
   \draw (3,0) to[short] (3,2);
   %vstup a vystup
   \draw (-2,1) to[short, o-*] (0,1);
   \draw (3,1) to[short, *-o] (5,1);
\end{circuitikz}
```

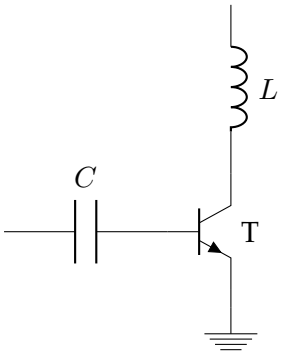
Tabulka 5.3 zobrazuje další schématické značky, které budeme v této části používat.

Značka	**Popis**	**Argument**
	kapacita	[capacitor] nebo [C]
	dioda	[empty diode] nebo [Do]
	dioda plná	[full diode] nebo [D*]
	indukce	[cute inductor] nebo [L]
	indukce US	[american inductor]

Značka	Popis	Argument
	tranzistor NPN	[npn]
	tranzistor PNP	[pnp]
	zem	[ground]

Tabulka 5.3: Schématické značky III

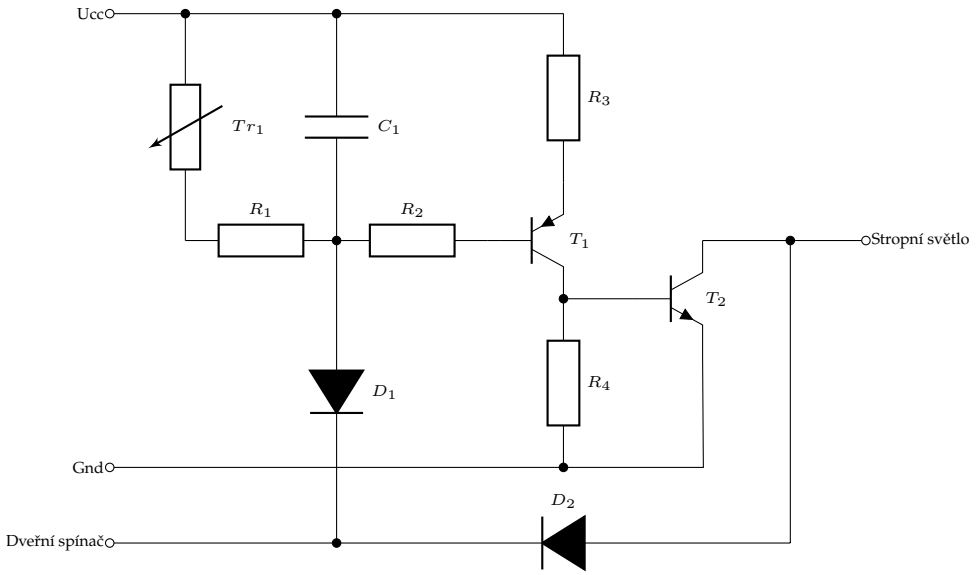
Na dalším obrázku můžeme zpozorovat vykreslení tranzistoru. Je důležité si jej pojmenovat pro pozdější připojování k jeho elektrodám.



Obrázek 5.7: Jednoduché schéma pro ukázkovou sazbu tranzistoru NPN

```
\begin{circuitikz}
   %vykresleni znacky tranzistoru (npn)
   \draw (0,0) node[npn] (npn) {} node[right]{T};
   %vykresleni znacky indukce L, pripojení ke kolektoru
   \draw (0,3) to[american inductor, l=$L$] (npn.C);
   %vykresleni znacky kapacity C, pripojení k bazi
   \draw (-3,0)  to[C, l=$C$] (npn.B);
   %vykresleni znacky zem, pripojení k emitoru
   \draw (0,-1) node[ground]{} to[short] (npn.E);
 \end{circuitikz}
```

Poslední obrázek 5.8 zobrazuje složitější schéma s použitím všech značek definovaných v tabulkách.

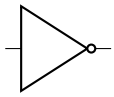
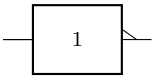
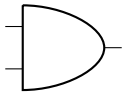
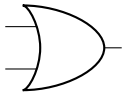
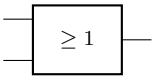
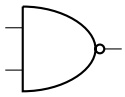
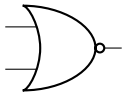
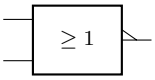
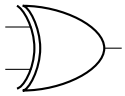
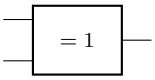
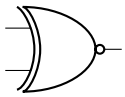
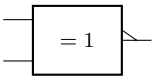


Obrázek 5.8: Složitější schéma

```
\begin{circuitikz}
   %vstup/výstup {Dverni spinac}
   \draw (0,0) node[left]{Dveřní spínač} to[short, o-](3,0);
   %vykresleni diody D1
   \draw (3,4) to[D*, l=$D_1$] (3,0);
   %vykresleni kapacity C1
   \draw (3,7) to[C, l=$C_1$] (3,4);
   \draw (3,7) -- (1,7);
   %vstup/vystup {UCC}
   \draw (1,7) to[short, -o] (0,7) node[left]{Ucc};
   %vykresleni rezistoru Tr s promennou hodnotou
   \draw (1,7) to[tgeneric, l=$Tr_1$, *-](1,4);
   %vykresleni rezistoru R1 a R2
   \draw (1,4) to[generic, l=$R_1$, -*] (3,4);
   \draw (3,4) to[generic, l=$R_2$] (5,4);
   %vykresleni tranzistoru T1
   \draw (6,4) node[pnp] (pnp) {} node[right] {$T_1$};
   \draw (pnp.B) -- (5,4);
   %vykresleni rezistoru R3
   \draw (3,7) to[short, *-] (6,7) to[generic=$R_3$] (pnp.E);
   %vykresleni tranzistoru T2
   \draw (pnp.C) node[npn, anchor=B, xshift=1cm] (npn) {} node[right=1.8cm] {$T_2$};
   \draw (pnp.C)--(npn.B);
   %vstup/vystup {Stropni svetlo } a {Gnd}
   \draw (npn.C) -- (8,4) to[short, -o] (10,4) node[right]{Stropní světlo};
   \draw (npn.E) -- (7.85,1) -- (6,1) to[short, -o] (0,1)node[left]{Gnd};
   %vykresleni rezistoru R4
   \draw (pnp.C) to[generic=$R_4$, *-*] (6,1);
   %vykresleni diody D2
   \draw (9,4) to[short, *-] (9,0) to[D*, l_=$D_2$, -*] (3,0);
\end{circuitikz}
```

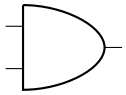
5.2 Logické obvody

Stavebním prvkem každého logického obvodu jsou hradla (logické členy). Balík `circuitikz` obsahuje všechny základní logická hradla. Tabulka 5.4 představuje schématické značky rozdělené podle normy. Dělíme je na značky dle americké normy ASA a schématické značky normy IEC.

Popis	Značka ASA	Argument	Značka IEC	Argument
NOT		[not port]		[european not port]
AND		[and port]		[european and port]
OR		[or port]		[european or port]
NAND		[nand port]		[european nand port]
NOR		[nor port]		[european nor port]
XOR		[xor port]		[european xor port]
XNOR		[xnor port]		[european xnor port]

Tabulka 5.4: Logická hradla - normy ASA/IEC

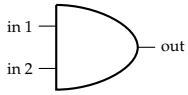
Všechny logické členy se vykreslují jako uzel, kde nejprve určíme bod umístění a poté do vlastnosti uzlu uvedeme argument hradla, který chceme vykreslit. Na obrázku 5.9 je patrné vykreslení hradla AND v bodě (0,0).



Obrázek 5.9: Vykreslení hradla AND

```
\begin{circuitikz}
   \draw (0,0) node[and port](and){};
\end{circuitikz}
```

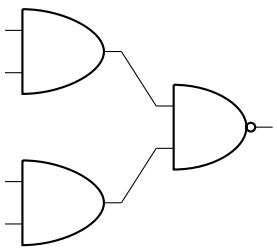
Každé hradlo má typicky jeden nebo více vstupů a jediný výstup. Abychom se mohli k těmto logickým členům připojovat, je třeba si každé hradlo pojmenovat. Porty pro vstup a výstup mají své specifické pojmenování. Pro vstupy se používá `in 1` (vstupní port 1) a `in 2` (vstupní port 2). Jelikož výstupní port je pouze jeden, tak se pro připojení používá `out`. Ke každému portu můžeme navíc přiřadit popisek (viz. obrázek 5.10).



Obrázek 5.10: Přidání popisků k jednotlivým portům

```
\begin{circuitikz}
   \draw (0,0) node[and port](and){}
     (and.in 1) node[left] {in 1} (and.in 2) node[left] {in 2}  (and.out) node[right] {out};
\end{circuitikz}
```

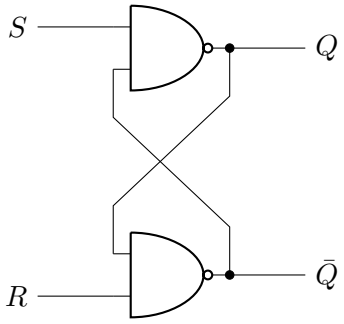
Na následujícím obrázku je patrné spojení několika logických členů. Pro spojení je možné použít `--` nebo `[short]`.



Obrázek 5.11: Spojení logických členů

```
\begin{circuitikz}
   %vykresleni 3 hradel
   \draw (0,0) node[and port](and) {};
   \draw (0,2) node[and port](and2) {};
   \draw (2,1) node[nand port](nand) {};
   %spojeni hradel
   \draw (and.out) -- (nand.in 2);
   \draw (and2.out) -- (nand.in 1);
\end{circuitikz}
```

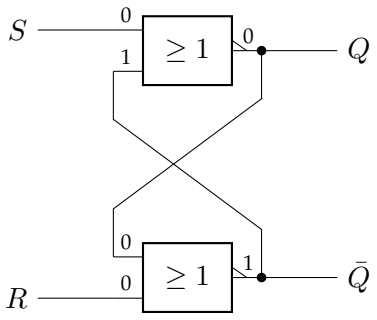
Na obrázku 5.12 je vytvořen klopný obvod RS sestavený z hradel NAND. Pro efektivní křížení čar jsou vytvořeny pomocné body. Popisky jsou od požadovaných portů vzdáleny o 1 cm (docílíme přehlednějšího zobrazení), které jsou následně spojeny se vstupy a výstupy hradla.



Obrázek 5.12: Klopný obvod RS realizovaný z hradel NAND

```
\begin{circuitikz}
   %vykresleni hradel
   \draw (0,0) node[nand port] (nandR) {};
   \draw (0,3) node[nand port] (nandS) {};

   %vytvoreni pomocnych bodu pro spojeni z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) node[above = 0.5cm](rIn1) {};
   \draw (nandR.out) node[above = 0.5cm](rOut) {};
   \draw (nandS.in 2) node[below = 0.5cm](sIn2) {};
   \draw (nandS.out) node[below = 0.5cm](sOut) {};

   %vykresleni car z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) -- (rIn1.center) -- (sOut.center) -- (nandS.out);
   \draw (nandS.in 2) -- (sIn2.center) -- (rOut.center) -- (nandR.out);

   %prodlouzeni vstupu a vystupu o 1 cm + pridani popisku
   \draw (nandR.in 2) node(b2)[left=1cm] {$R$} -- (b2.east);
   \draw (nandS.in 1) node(b1)[left=1cm] {$S$} -- (b1.east);
   \draw (nandS.out) node(b3)[right=1cm]{$Q$} to[short, *-] (b3.west);
   \draw (nandR.out) node(b4)[right=1cm]{$ \bar Q$} to[short, *-](b4.west);
\end{circuitikz}
```

Následující obrázek slouží pro ukázku vkládání popisků k jednotlivým portům. Musíme si nejdříve určit port ke kterému přidáme popisek a následně zvolit stranu pro vykreslení.

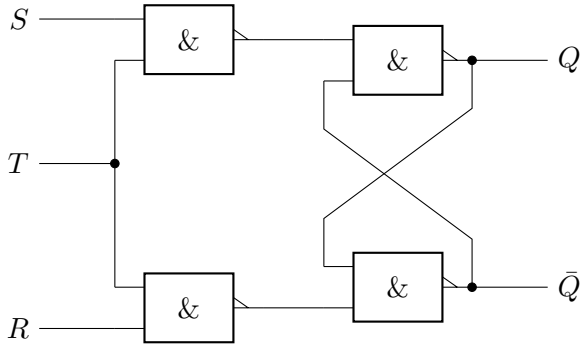


Obrázek 5.13: Klopný obvod RS realizovaný z hradel NOR; přidány popisky k portům

```
\begin{circuitikz}
   %vykresleni hradel
   \draw (0,0) node[european nor port] (norR) {};
   \draw (0,3) node[european nor port] (norS) {};

   %vytvoreni pomocnych bodu pro spojeni z jednoho hradla do druheho a naopak
   \draw (norR.in 1) node[above = 0.5cm](rIn1) {};
   \draw (norR.out) node[above = 0.5cm](rOut) {};
   \draw (norS.in 2) node[below = 0.5cm](sIn2) {};
   \draw (norS.out) node[below = 0.5cm](sOut) {};

   %vykresleni car z jednoho hradla do druheho a naopak
   \draw (norR.in 1) -- (rIn1.center) -- (sOut.center) -- (norS.out);
   \draw (norS.in 2) -- (sIn2.center) -- (rOut.center) -- (norR.out);

   %prodlouzeni vstupu a vystupu o 1 cm + pridani popisku
   \draw (norR.in 2) node(b2)[left=1cm] {$R$} -- (b2.east);
   \draw (norS.in 1) node(b1)[left=1cm] {$S$} -- (b1.east);
   \draw (norS.out) node(b3)[right=1cm]{$Q$}   to[short, *-](b3.west);
   \draw (norR.out) node(b4)[right=1cm]{$ \bar Q$} to[short, *-](b4.west);

   %pridani popisku 0 a 1 k portum
   \scriptsize
   \draw (norS.in 1) node[above right]{0}
        (norS.in 2) node[above right]{1}
        (norR.in 1) node[above right]{0}
        (norR.in 2) node[above right]{0}
        (norS.out) node[above left]{0}
        (norR.out) node[above left]{1};
\end{circuitikz}
```

Poslední dva obrázky vykreslují paměťové členy RST a D. Základem obou těchto členů je klopný obvod RS.

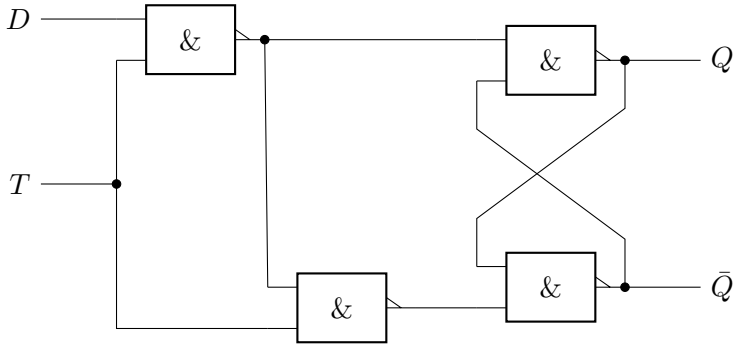
Vykreslení paměťového členu RST



Obrázek 5.14: Paměťový člen RST

```
\begin{circuitikz}
   %vykresleni hradel
   \draw (0,0) node[european nand port] (nandR) {};
   \draw (0,3) node[european nand port] (nandS) {};

   %vytvoreni pomocnych bodu pro spojeni z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) node[above = 0.5cm](rIn1) {};
   \draw (nandR.out) node[above = 0.5cm](rOut) {};
   \draw (nandS.in 2) node[below = 0.5cm](sIn2) {};
   \draw (nandS.out) node[below = 0.5cm](sOut) {};

   %vykresleni car z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) -- (rIn1.center) -- (sOut.center) -- (nandS.out);
   \draw (nandS.in 2) -- (sIn2.center) -- (rOut.center) -- (nandR.out);

   %vyvoreni hradel 1 a 2, ktere jsou pripojeny na vstupy S a R
   \draw (nandR.in 2) node[european nand port, left=1cm](nand2){} -- (nand2.out);
   \draw (nandS.in 1) node[european nand port, left=1cm](nand1){} -- (nand1.out);

   %vytvoreni synchronizacniho signalu T
   \draw (nand1.in 2) -- (nand2.in 1) node[above = 1.5cm](b0){};
   \draw (b0.center) node(Tin)[left=1cm]{$T$} to[short, *-] (Tin.east);

   %prodlouzeni vstupu a vystupu o 1 cm + pridani popisku
   \draw (nand2.in 2) node(b2)[left=1cm] {$R$} -- (b2.east);
   \draw (nand1.in 1) node(b1)[left=1cm] {$S$} -- (b1.east);
   \draw (nandS.out) node(b3)[right=1cm]{$Q$}   to[short, *-](b3.west);
   \draw (nandR.out) node(b4)[right=1cm]{$ \bar Q$} to[short, *-](b4.west);
\end{circuitikz}
```

Vykreslení paměťového členu D



Obrázek 5.15: Paměťový člen D

```
\begin{circuitikz}
   %vykresleni hradel
   \draw (0,0) node[european nand port] (nandR) {};
   \draw (0,3) node[european nand port] (nandS) {};

   %vytvoreni pomocnych bodu pro spojeni z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) node[above = 0.5cm](rIn1) {};
   \draw (nandR.out) node[above = 0.5cm](rOut) {};
   \draw (nandS.in 2) node[below = 0.5cm](sIn2) {};
   \draw (nandS.out) node[below = 0.5cm](sOut) {};

   %vykresleni car z jednoho hradla do druheho a naopak
   \draw (nandR.in 1) -- (rIn1.center) -- (sOut.center) -- (nandS.out);
   \draw (nandS.in 2) -- (sIn2.center) -- (rOut.center) -- (nandR.out);

   %vyvoreni hradel 1 a 2, ktere jsou pripojeny na vstupy S a R
   \draw (nandR.in 2) node[european nand port, left=1cm](nand2){} -- (nand2.out);
   \draw (nandS.in 1) node[european nand port, left=3cm](nand1){} -- (nand1.out);
   \draw (nand2.in 1) to[short, -*] (nand1.out);
   %vytvoreni synchronizacniho signalu T
   \draw (nand2.in 2) -| (nand1.in 2) node[below = 1.5cm](b0){};
   \draw (b0.center) node(Tin)[left=1cm]{$T$} to[short, *-] (Tin.east);

   %prodlouzeni vstupu a vystupu o 1 cm + pridani popisku
   \draw (nand1.in 1) node(b1)[left=1cm] {$D$} -- (b1.east);
   \draw (nandS.out) node(b3)[right=1cm]{$Q$}   to[short, *-](b3.west);
   \draw (nandR.out) node(b4)[right=1cm]{$ \bar Q$} to[short, *-](b4.west);
\end{circuitikz}
```

6 Grafy naměřených hodnot a matematických funkcí

V kapitole 2 jsme si ukázali způsob vytvoření jednoduchého grafu, který byl vykreslen za pomocí dvou přímek znázorňující osy X a Y. Na tyto osy jsme po použití příkazu `foreach` vykreslili požadované body. Pro lepší orientaci ve výsledném obrázku byl graf taktéž doplněn o mřížku. Toto krátké připomenutí nám poslouží jako základ k této kapitole.

Kapitola se bude dělit do dvou částí, kde první část se věnuje grafickému znázornění naměřených hodnot načtených z externího souboru, zatímco druhá část této kapitoly se zabývá matematickými funkcemi a jejím zobrazením v grafu.

6.1 Grafy naměřených hodnot

V této části si ukážeme jednu z možností ukládání hodnot do souboru CSV. Obsahem tohoto souboru budou tedy záznamy, které následně zobrazíme v tabulce a v grafu.

Pro uložení hodnot do souboru CSV se používá prostředí `{filecontents}`. Toto prostředí je ohraničeno dvojicí příkazů `\begin{filecontents}` a `\end{filecontents}`. Hodnoty se mezi těmito příkazy vkládají po řádcích, kde se jednotlivé položky oddělují čárkou. Pokud text některé z položek obsahují čárku, je možné celý tento text zaobalit do uvozovek. Toto prostředí je nutné vložit do preambule.

V našem případě máme uloženy hodnoty do několika řádků, kde každý řádek obsahuje čtyři položky, které jsou odděleny čárkou. Tím nám na první pohled vznikla jakási tabulka, která se skládá ze čtyř sloupců a několika řádků. První řádek nám bude reprezentovat hlavičku tabulky. Pro úplnost je nutné do složených závorek zadat název souboru s příponou .csv do kterého se celý tento obsah uloží.

```
\begin{filecontents*}{hodnoty.csv}
   a,   b,   c,   d
   1,   400, 550, 250
   2,   550, 450, 400
   3,   450, 300, 400
   4,   600, 400, 450
   5,   550, 450, 350
   6,   600, 450, 400
   7,   500, 550, 300
   8,   650, 500, 400
   9,   700, 500, 350
   10,  600, 400, 500
   11,  700, 600, 450
   12,  750, 600, 500
\end{filecontents*}
\begin{document}
   ...
\end{document}
```

Po úspěšném vytvoření souboru si jeho obsah vygenerujeme do tabulky. Pro tuto akci můžeme použít balíček `csvsimple` [4], který si musíme nejdříve naimportovat do preambule. Balíček podporuje mnoho způsobů jak toho dosáhnout.

Asi nejednoduší varianta je použití příkazu `\csvautotabular{název souboru}`, který automaticky vygeneruje celou tabulku. V podstatě to co jsme do souboru zapsali se nám vygeneruje do tabulky. Tento způsob je vhodný v případě, kdy již nepotřebujeme tabulku nijak dále upravovat.

a	b	c	d
1	400	550	250
2	550	450	400
3	450	300	400
4	600	400	450
5	550	450	350
6	600	450	400
7	500	550	300
8	650	500	400
9	700	500	350
10	600	400	500
11	700	600	450
12	750	600	500

Tabulka 6.1: Automatické generování tabulky - použití příkazu `csvautotabular`

```
\csvautotabular{hodnoty.csv}
```

V případě kdy chceme mít absolutní kontrolu nad danými záznamy, použijeme klasickou metodu pro tvorbu tabulky použitím prostředí `tabular`. Balíček `csvsimple` obsahuje příkaz `csvreader`, který nám umožní čtení v souboru CSV. V parametru tohoto příkazu poté definujeme argument `late after line`, který se volá na konci každého řádku. V našem případě se provede akce odřádkování. V dalším kroku musíme uvést jméno souboru, ze kterého chceme data načítat. Poté následuje dvojice uzavřených složených závorek. V první dvojici přiřadíme ke každému sloupci jméno proměnné, zatímco druhá dvojice data uložená v proměnné vypisuje do tabulky.

Měsíc	2010	2011	2012
1	400	550	250
2	550	450	400
3	450	300	400
4	600	400	450
5	550	450	350
6	600	450	400
7	500	550	300
8	650	500	400
9	700	500	350
10	600	400	500
11	700	600	450
12	750	600	500

Tabulka 6.2: Ukládání záznamu do tabulky - použití prostředí `tabular`

```
\begin{tabular}{|c|c|c|c|}
   \hline
    Měsíc & 2010 & 2011 & 2012\\
   \hline
   \csvreader[late after line=\\]{hodnoty.csv}
    {1=\a, 2=\b, 3=\c, 4=\d}
    {\a & \b & \c & \d}
   \hline
\end{tabular}
```

Jiná varianta pro vykreslení tabulky spočívá v použití samotného příkazu `csvreader` bez podpory prostředí `tabular`. Ve vlastnostech tohoto příkazu si nadefinujeme takové argumenty, které nám umožní vytvořit tabulku, kterou požadujeme. Argumenty je třeba od sebe oddělovat čárkou. Prvním argumentem, který je třeba uvést, se nazývá `tabular`. Už z názvu vyplývá, že se jedná o argument zajišťující vykreslení tabulky. Dalším parametrem je pak `table head`, jenž zajišťuje text uvedený v hlavičce. Argument `late after line` se volá na konci každého řádku, zatímco argument `late after last line` je zavolán na konci posledního řádku.

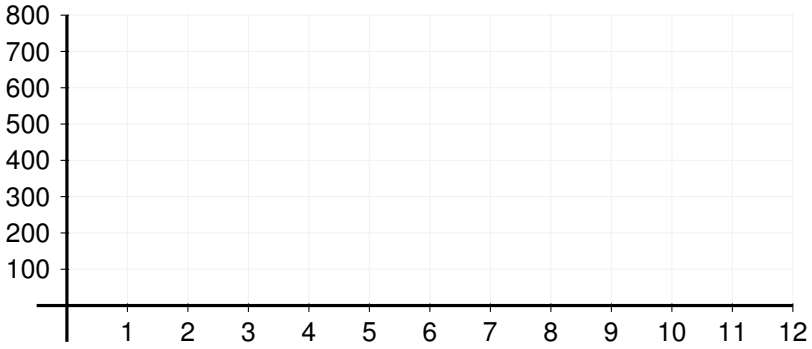
Měsíc	2010	2011	2012
1	400	550	250
2	550	450	400
3	450	300	400
4	600	400	450
5	550	450	350
6	600	450	400
7	500	550	300
8	650	500	400
9	700	500	350
10	600	400	500
11	700	600	450
12	750	600	500

Tabulka 6.3: Ukládání záznamu do tabulky - bez použití prostředí `tabular`

```
\csvreader
   [tabular=|c|c|c|c|,
   table head=
    \hline
    Měsíc & 2010 & 2011 & 2012 \\
    \hline
    \hline,
   late after line=\\,
   late after last line=\\ \hline]{hodnoty.csv}
   {1=\a,2=\b,3=\c,4=\d}
   {\a & \b & \c & \d}
```

Zatím jsme hodnoty uložené v externím souboru vypsali do tabulky. Nyní si ukážeme, jak tyto záznamy vykreslit do grafu. Na obrázku 6.1 jsme si vytvořili jednoduchý graf. Osa X bude představovat první sloupec tabulky. Za pomocí příkazu `foreach` vykreslíme body (1,...,12) na osu X. Na osu Y budeme nanášet hodnoty ze sloupců b, c, d. Jelikož nejvyšší hodnota uložená v tabulce je menší než 800, tak body budeme vykreslovat na osu Y za pomocí příkazu `foreach` v

rozmezí (100,...,800). Abychom použili tohoto rozmezí, je nutné změnit měřítko svislé osy. Toto nastavení se provede ve vlastnostech prostředí `tikzpicture`. Ve výchozím stavu je měřítko pro osy nastaveno na hodnotu 1 cm. Abychom dosáhli požadovaného výsledku, je třeba změnit měřítko pro osu Y na hodnotu 0.06 mm. Toto měřítko se poté aplikuje v celém prostředí.

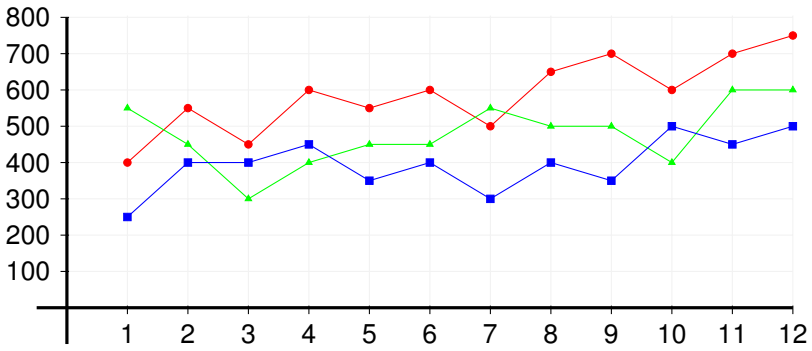


Obrázek 6.1: Automatické generování tabulky

```
\begin{tikzpicture}[y=0.06mm, x=1cm, font=\sffamily]
   %mrizka
   \draw [gray!10, xstep= 1cm, ystep=0.6cm ](0,0) grid (12,805);

   %osy
   \draw [very thick](-0.5,0) -- coordinate (x axis mid) (12,0);
   \draw [very thick](0,-100) -- coordinate (y axis mid) (0,800);

   %vykresleni bodu na osach
   \foreach \x in {1,2,...,12}
      \draw (\x,1pt) -- (\x,-3pt) node[anchor=north] {\x};
   \foreach \y in {100,200,...,800}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};
\end{tikzpicture}
```

Nyní obrázek 6.1 doplníme o údaje, které máme uložené v souboru. Pro čtení v souboru opět použijeme příkaz `csvreader`. Do vlastností příkazu uvedeme několik argumentů. Definice argumentu `head to column names` způsobí, že zápisy uvedeny v hlavičce (a, b, c nebo d) se automaticky použijí jako proměnné. Argument `late after head` se zavolá pro první řádek, jenž následuje po řádku reprezentující hlavičku. Poslední argument, který je třeba zmínit se nazývá `after line`. Tento parametr se zavolá pro každý řádek v souboru. Akce prováděné v těchto dvou argumentech, předávají hodnotu proměnné (a, b, c nebo d) do námi vytvořeného příkazu (aOld, bOld, cOld nebo dOld). Hodnoty uložené v těchto proměnných a příkazech se poté použijí na samotné vykreslení křivek (datové řady). Pro každou křivku zvolíme jiný symbol značení uzlů. Tohoto výsledku docílíme použitím příkazu `pgfuseplotmark`, kde do složených závorek uvedeme požadovaný symbol. Pro použití tohoto příkazu je nutné do preambule zavést knihovnu `plotmarks`.

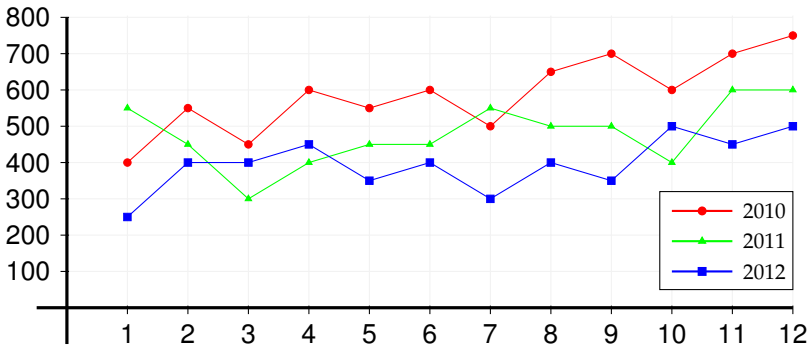
```
\usetikzlibrary{plotmarks}
```



Obrázek 6.2: Automatické generování tabulky

```
\begin{tikzpicture}[y=0.06mm, x=1cm, font=\sffamily]
  %mrizka
  \draw [gray!10, xstep= 1cm, ystep=0.6cm ](0,0) grid (12,805);

  %osy
  \draw [very thick](-0.5,0) -- coordinate (x axis mid) (12,0);
  \draw [very thick](0,-100) -- coordinate (y axis mid) (0,800);

  %vykresleni bodu na osach
  \foreach \x in {1,2,...,12}
     \draw (\x,1pt) -- (\x,-3pt) node[anchor=north] {\x};
  \foreach \y in {100,200,...,800}
     \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

  %nacteni hodnot ze souboru
  \csvreader[
     head to column names,
     late after head=
        \xdef\aOld{\a}
        \xdef\bOld{\b}
        \xdef\cOld{\c}
        \xdef\dOld{\d},
     after line=
        \xdef\aOld{\a}
        \xdef\bOld{\b}
        \xdef\cOld{\c}
        \xdef\dOld{\d}]{hodnoty.csv}{}

     %vykresleni krivek
     {
       \draw[red] (\aOld, \bOld) -- (\a,\b) node[scale=0.7] {$\pgfuseplotmark{*}$};
       \draw[green] (\aOld, \cOld) -- (\a,\c) node[scale=0.7] {$\pgfuseplotmark{triangle*}$};
       \draw[blue] (\aOld, \dOld) -- (\a,\d) node[scale=0.7] {$\pgfuseplotmark{square*}$};
     }
\end{tikzpicture}
```

V tomto okamžiku je graf zcela hotov. Nicméně je zapotřebí křivky nějakým způsobem pojmenovat. Doplníme obrázek 6.2 o legendu, jenž zajistí snadnou orientaci v grafu. Vytvoříme si prostředí `scope`, do kterého následně vykreslíme křivky, které budou spojeny se jménem datové řady. Celé toto prostředí umístíme na místo, které nebude křivky překrývat, a to použitím argumentu `shift`.



Obrázek 6.3: Automatické generování tabulky

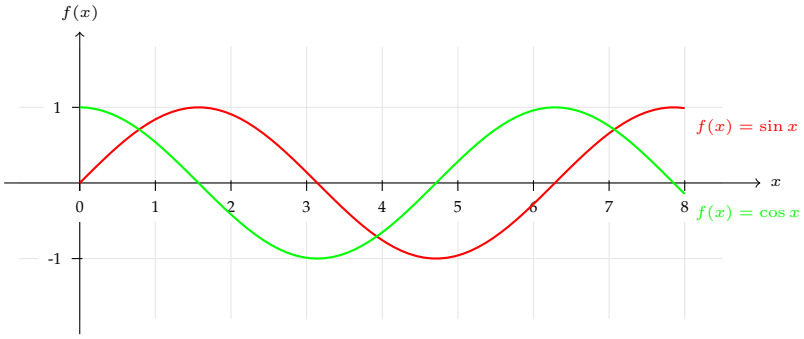
```
\begin{tikzpicture}[y=0.06mm, x=1cm, font=\sffamily]
  %mrizka
  \draw [gray!10, xstep= 1cm, ystep=0.6cm ](0,0) grid (12,805);
  %osy
  \draw [very thick](-0.5,0) -- coordinate (x axis mid) (12,0);
  \draw [very thick](0,-100) -- coordinate (y axis mid) (0,800);
  %vykresleni bodu na osach
  \foreach \x in {1,2,...,12}
     \draw (\x,1pt) -- (\x,-3pt) node[anchor=north] {\x};
  \foreach \y in {100,200,...,800}
     \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};
  %nacteni hodnot ze souboru
  \csvreader[
     head to column names,
     late after head=
        \xdef\aOld{\a}
        \xdef\bOld{\b}
        \xdef\cOld{\c}
        \xdef\dOld{\d},
     after line=
        \xdef\aOld{\a}
        \xdef\bOld{\b}
        \xdef\cOld{\c}
        \xdef\dOld{\d}]{hodnoty.csv}{}

     %vykresleni krivek
     {
       \draw[red] (\aOld, \bOld) -- (\a,\b) node[scale=0.7] {$\pgfuseplotmark{*}$};
       \draw[green] (\aOld, \cOld) -- (\a,\c) node[scale=0.7] {$\pgfuseplotmark{triangle*}$};
       \draw[blue] (\aOld, \dOld) -- (\a,\d) node[scale=0.7] {$\pgfuseplotmark{square*}$};
     }

  %legenda
  \begin{scope}[shift={(10,100)}]
     \draw[fill=white] (-0.2,-50) rectangle (2,220);
     \draw[draw = blue, thick] (0,0) -- (0.5cm,0)
          node[blue] {$\pgfuseplotmark{square*}$} -- (1cm,0) node[right]{2012};
     \draw[draw = green, thick, yshift=0.5cm] (0,0) -- (0.5cm,0)
          node[green] {$\pgfuseplotmark{triangle*}$} -- (1cm,0) node[right]{2011};
     \draw[draw = red, thick, yshift=1cm] (0,0) -- (0.5cm,0)
          node[red] {$\pgfuseplotmark{*}$} -- (1cm,0) node[right]{2010};
  \end{scope}
\end{tikzpicture}
```

6.2 Grafy matematických funkcí

V této podkapitole si ukážeme vykreslení obrázků znázorňující grafy jednoduchých matematických funkcí. Pro vkládání matematických výrazů bylo do jazyka PGF implementováno matematické jádro, které umožní při správné syntaxi za pomocí balíku Ti*k*Z vykreslit požadovanou funkci v grafu. Existuje více možností grafického znázornění matematických funkcí. Nejednodušší variantou, kterou budeme v této části používat je vypsání předpisu funkce. Pro její zadání je možné využít několik předdefinovaných matematických příkazů např. *sin*, *cos*, *tan*, *cot*, *ln*, *exp*, *sqrt* a další.

Z obrázku 6.4 můžeme zpozorovat grafické zobrazení dvou funkcí. Pro vykreslení funkce v grafu použijeme klíčové slovo `plot`. Za tímto klíčovým slovem se nachází kulaté závorky udávající polohu bodu o souřadnicích (x, y), kde $y = f(x)$. Parametr x bude sloužit jako proměnná $\x$, do které se budou postupně vkládat hodnoty z intervalu, který uvedeme ve vlastnostech prostředí `tikzpicture` argumentem `domain`. V našem případě se budou funkce vykreslovat v intervalu od 0 do 8. Argumentem `samples` udáváme z kolika bodů se bude funkce v tomto intervalu skládat. Za parametr y dosadíme ve složených závorkách danou funkci.

Následující příklad vyobrazuje graf matematických funkcí sinu a cosinu.



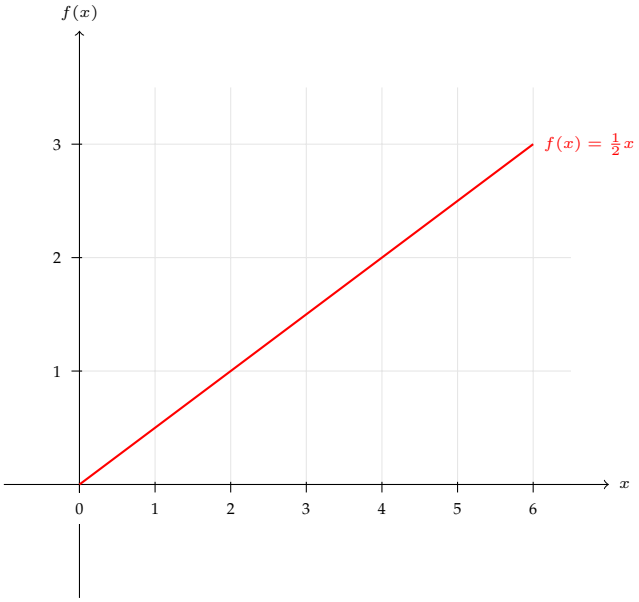
Obrázek 6.4: Graf funkce sinus a cosinus

```
\begin{tikzpicture}[domain=0:8, samples=80, font=\tiny]
   %mrizka
   \draw[gray!20] (-0.8,-1.8) grid (8.5,1.8);
   %vykresleni os
   \draw[->] (-1,0) -- (9,0) node[right] {$x$};
   \draw[->] (0,-2) -- (0,2) node[above] {$f(x)$};
   %vykresleni bodu na osach
   \foreach \x in {0,1,...,8}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {-1,1}
      \draw (1pt,\y) -- (-3pt,\y) node[fill=white, anchor=east] {\y};

   %vykresleni funkci
   \draw [color=red, thick] plot (\x,{sin(\x r)})
      node[below right] {$f(x) = \sin x$};
   \draw [color=green, thick] plot (\x,{cos(\x r)})
      node[below right] {$f(x) = \cos x$};
\end{tikzpicture}
```

Následující grafy slouží pro pochopení, jakým způsobem můžeme vykreslit jednoduché funkce za pomoci matematických příkazů. Tyto příkazy nám mohou přijít vhod v případě, kdy chceme vyobrazit funkci z daleko složitějším předpisem. Na minulém obrázku jsme si ukazovali vypsání intervalu do vlastností prostředí `tikzpicture`. To je vhodné především v případě vykreslení jedné funkce v grafu, popř. více funkcí využívající stejného intervalu. Pro zbylé grafy použijeme variantu, kdy interval umístíme do vlastností příkazu `plot`. Tato varianta se hodí v situaci, kdy každé funkci přiřadíme jiný interval.

Obrázek 6.5 znázorňující vykreslení grafu lineární funkce v intervalu (0 – 6) a s předpisem:

$$f(x) = \tfrac{1}{2}x$$

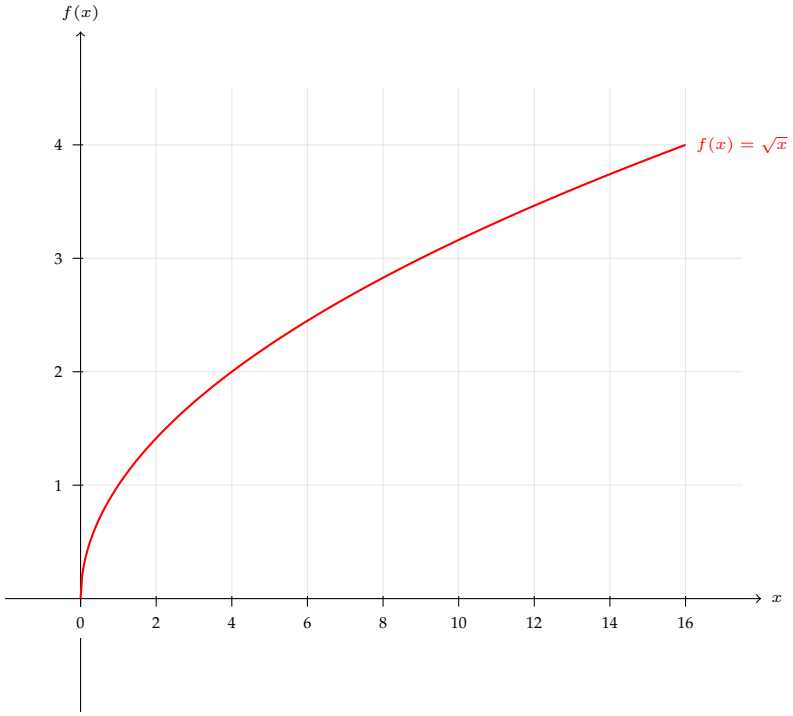


Obrázek 6.5: Graf lineární funkce

```
\begin{tikzpicture}[y=1.5cm, x=1cm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.5cm] (0,0) grid (6.5,4.5);

   %vykresleni os
   \draw[->] (-1,0) -- (7,0) node[right] {$x$};
   \draw[->] (0,-1) -- (0,5) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {0,1,...,6}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {1,2,3,4}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick] plot [domain=0:6] (\x , {1/2*(\x)})
      node[right, fill=white] {$f(x) = \frac{1}{2} x$};
\end{tikzpicture}
```

Obrázek 6.6 znázorňující vykreslení grafu druhé odmocniny v intervalu (0 – 16) a s předpisem:

$$f(x) = \sqrt{x}$$

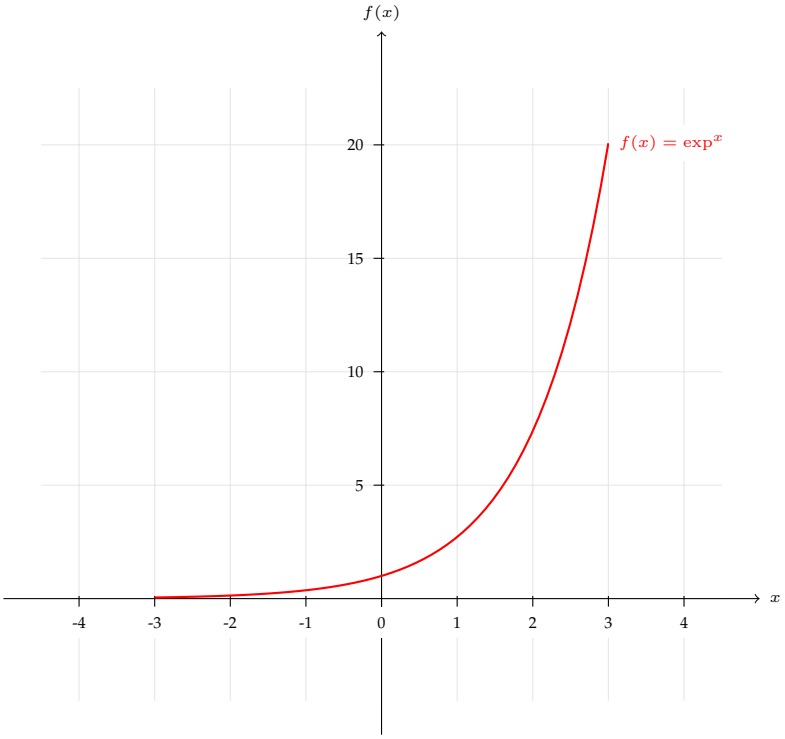


Obrázek 6.6: Graf druhé odmocniny

```
\begin{tikzpicture}[y=1.5cm, x=0.5cm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.5cm] (0,0) grid (17.5,4.5);

   %vykresleni os
   \draw[->] (-2,0) -- (18,0) node[right] {$x$};
   \draw[->] (0,-1) -- (0,5) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {0,2,4,...,16}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {1,2,3,4}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick] plot[domain=0:16, samples=400] (\x , {sqrt(\x)})
     node[right, fill=white] {$f(x) = \sqrt x$};
\end{tikzpicture}
```

Obrázek 6.7 znázorňující vykreslení grafu exponenciální funkce v intervalu (-3 – 3) a s předpisem:

$$f(x) = \exp^x$$

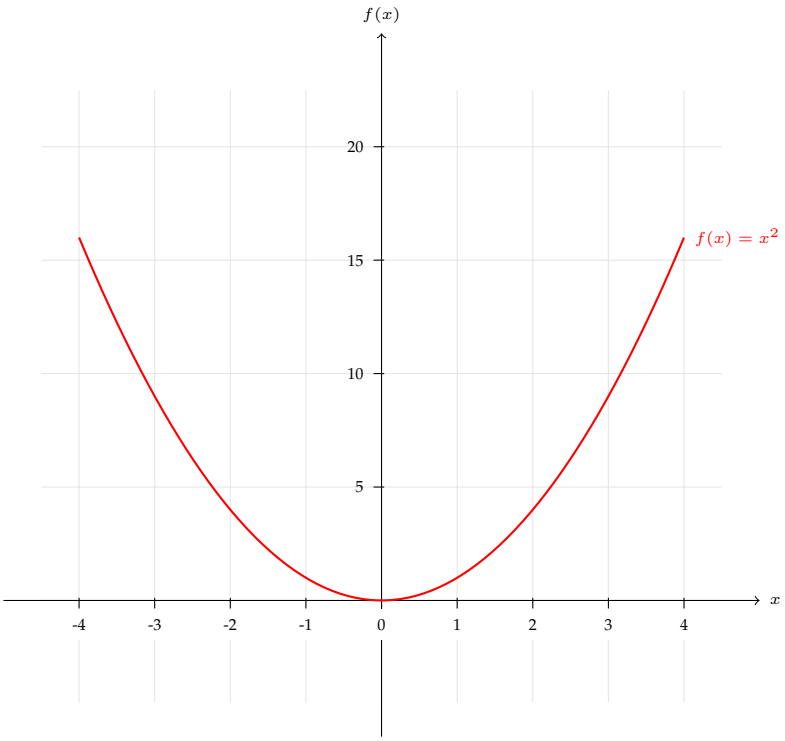


Obrázek 6.7: Graf exponenciální funkce

```
\begin{tikzpicture}[y=3mm, x=1cm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.5cm] (-4.5,-4.5) grid (4.5,22.5);

   %vykresleni os
   \draw[->] (-5,0) -- (5,0) node[right] {$x$};
   \draw[->] (0,-6) -- (0,25) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {-4,-3,-2,-1,0,1,2,3,4}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {5,10,...,20}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick] plot [domain=-3:3, samples=60] (\x , {exp(\x)})
      node[right, fill=white] {$f(x) = \exp^x$};
\end{tikzpicture}
```

Obrázek 6.8 znázorňující vykreslení grafu druhé mocniny v intervalu (-4 – 4) a s předpisem:

$$f(x) = x^2$$

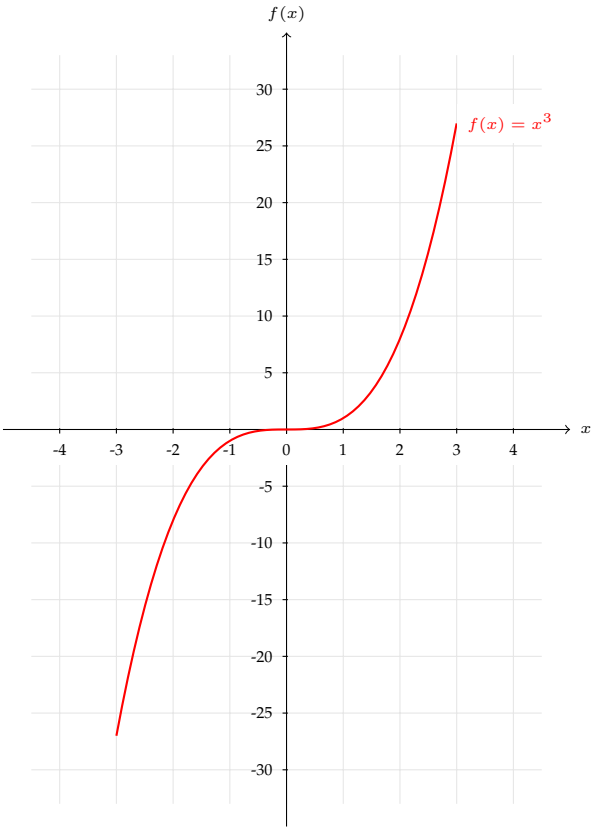


Obrázek 6.8: Graf druhé mocniny

```
\begin{tikzpicture}[y=3mm, x=1cm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.5cm] (-4.5,-4.5) grid (4.5,22.5);

   %vykresleni os
   \draw[->] (-5,0) -- (5,0) node[right] {$x$};
   \draw[->] (0,-6) -- (0,25) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {-4,-3,-2,-1,0,1,2,3,4}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {5,10,...,20}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivek
   \draw[color=red, thick] plot [domain=-4:4, samples=80] (\x , {(\x)^2})
      node[right, fill=white] {$f(x) = x^2$};
\end{tikzpicture}
```

Obrázek 6.9 znázorňující vykreslení grafu třetí mocniny v intervalu (-3 – 3) a s předpisem:

$$f(x) = x^3$$

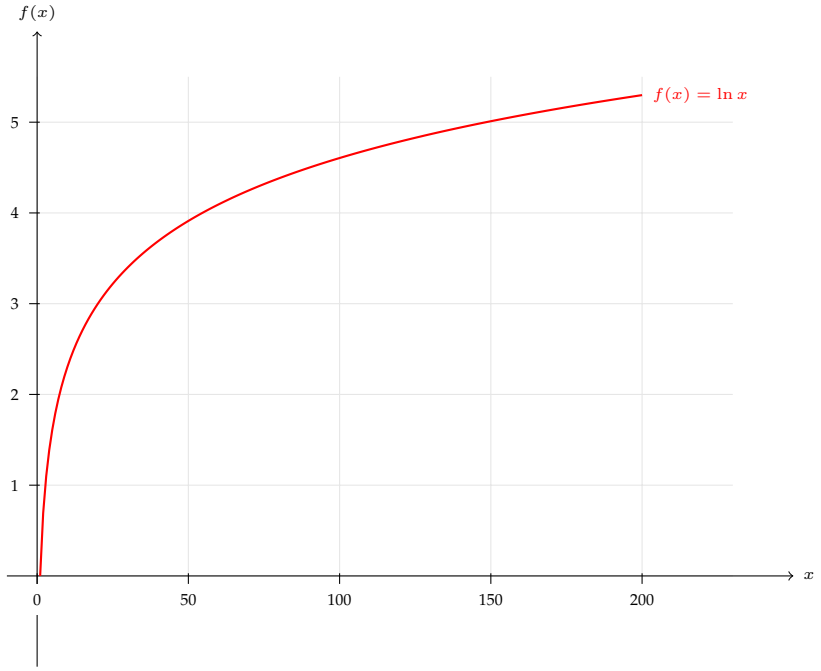


Obrázek 6.9: Graf třetí mocniny

```
\begin{tikzpicture}[y=3mm, x=1.5cm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.5cm, xstep=1.5cm] (-4.5,-33) grid (4.5,33);

   %vykresleni os
   \draw[->] (-5,0) -- (5,0) node[right] {$x$};
   \draw[->] (0,-35) -- (0,35) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {-4,-3,-2,-1,0,1,2,3,4}
      \draw (\x,1pt) -- (\x,-3pt) node[fill=white, anchor=north] {\x};
   \foreach \y in {-30,-25,...,-5,5,10,...,30}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick] plot [domain=-3:3, samples=60] (\x , {(\x)^3})
      node[right, fill=white] {$f(x) = x^3$};
\end{tikzpicture}
```

Obrázek 6.10 znázorňující vykreslení grafu logaritmické funkce v intervalu (1 – 200) a s předpisem:

$$f(x) = \ln x$$

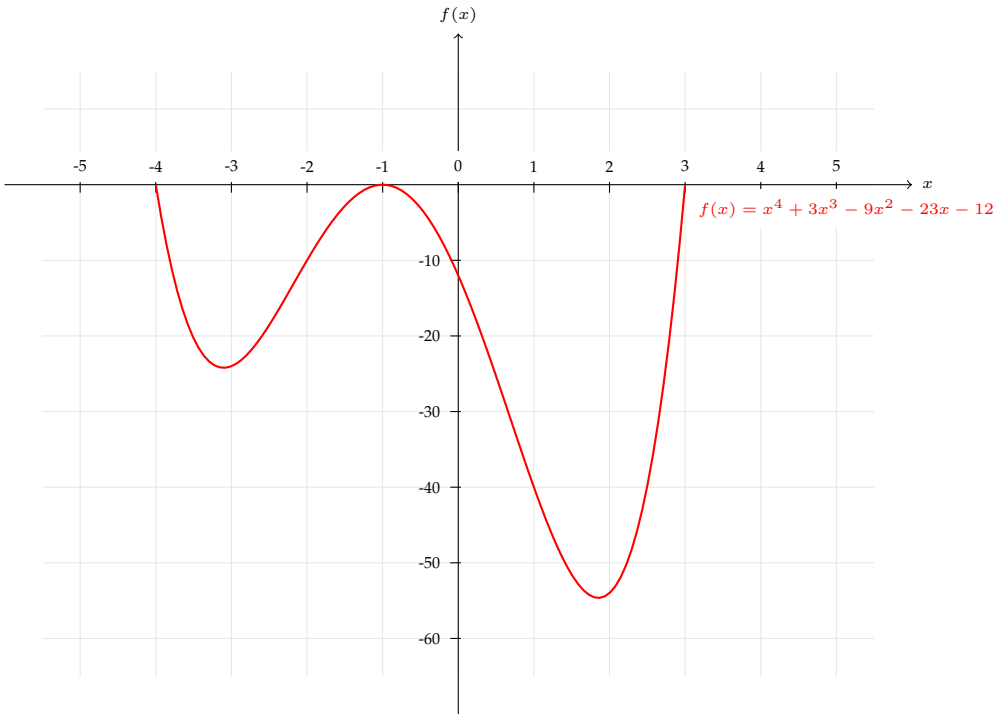


Obrázek 6.10: Graf logaritmické funkce

```
\begin{tikzpicture}[y=1.2cm, x=0.4mm, font=\tiny]
   %mrizka
   \draw[gray!20, ystep=1.2cm, xstep=2cm] (0,0) grid (230,5.5);

   %vykresleni os
   \draw[->] (-10,0) -- (250,0) node[right] {$x$};
   \draw[->] (0,-1) -- (0,6) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {0,50,100,150,200}
      \draw (\x,1pt) -- (\x,-3pt) node[anchor=north] {\x};
   \foreach \y in {1,2,3,4,5}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick] plot [domain=1:200, samples=200] (\x , {ln \x})
         node[right, fill=white] {$f(x) = \ln x$};
\end{tikzpicture}
```

Obrázek 6.11 znázorňující vykreslení grafu polynomické funkce v intervalu (-4 – 3) a s předpisem::

$$f(x) = x^4 + 3x^3 - 9x^2 - 23x - 12.$$



Obrázek 6.11: Graf polynomické funkce

```
\begin{tikzpicture}[y=1mm, x=1cm, font=\tiny]
   %mrizka
   \draw[very thin,color=gray!20] (-5.5,-65) grid (5.5,15);

   %vykresleni os
   \draw[->] (-6,0) -- (6,0) node[right] {$x$};
   \draw[->] (0,-70) -- (0,20) node[above] {$f(x)$};

   %vykresleni bodu na osach
   \foreach \x in {-5,-4,...,5}
      \draw (\x,-3pt) -- (\x,1pt) node[anchor=south, fill=white] {\x};
   \foreach \y in {-10,-20,...,-60}
      \draw (1pt,\y) -- (-3pt,\y) node[anchor=east] {\y};

   %vykresleni krivky
   \draw[color=red, thick]
      plot [domain=-4:3, samples=100] (\x , {(1*(\x)^4)+(3*(\x)^3)-(9*(\x)^2)-(23*\x)-12})
        node[below right=0.5mm, fill=white] {$f(x) = x^4+3x^3-9x^2-23x-12$};
\end{tikzpicture}
```

7 Prezentace a animace

V této kapitole si ukážeme tvorbu prezentací a animací, obsahující obrázky vytvořené pomocí balíčku Ti*k*Z.

7.1 Prezentace

Abychom mohli vytvářet prezentace, je nutné použít třídu `beamer` [7], která se aplikuje na místo klasických tříd typu `article` popř. `book`.

```
\documentclass{beamer}
```

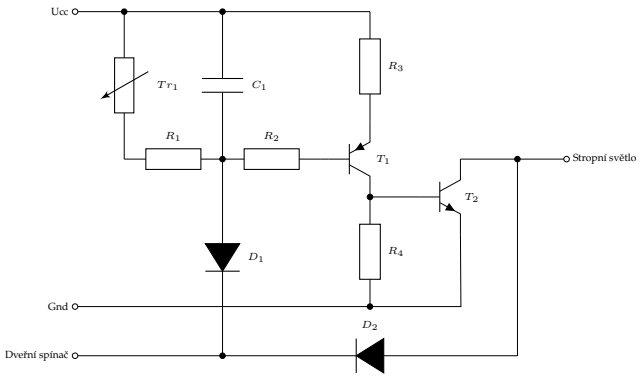
Příkaz `\usecolortheme` slouží pro definici šablony, která zajistí vzhled prezentace (barevný styl).

```
\usetheme{Warsaw}
```

Pokud budeme chtít prezentaci doplnit o titulní název, jméno autora a datum, použijeme k tomu příkazy `\title`, `\author` a `\date`. Od tohoto okamžiku můžeme do dokumentu postupně přidávat jednotlivé slidy.

Pro vytvoření slidu se používá prostředí `frame`, složené z dvojicí příkazů `\begin{frame}` a `\end{frame}`. Veškerý obsah, který chceme ve slidu zobrazit, se musí nacházet uvnitř tohoto prostředí. Do tohoto obsahu můžeme navíc definovat příkaz `\frametitle`, který doplní slide o nadpis. Způsob, jakým se nadpis zobrazí, záleží na zvolené šabloně. Za pomocí příkazu `\tableofcontents` si můžeme ze sekcí a podsekcí definovaných v prezentaci vygenerovat obsah.

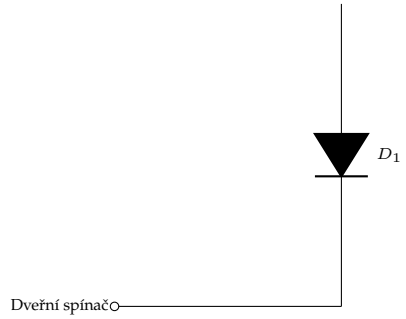
Nyní si ukážeme, jak prezentaci doplnit o obrázek, který jsme si již dříve vytvořili. Na něj následně aplikujeme dvě metody, které nám zajistí postupné odkrývání jeho částí. Zdrojový kód k tomuto obrázku nalezneme v kapitole 5 (obrázek 5.8).



Obrázek 7.1: Obrázek použitý v prezentaci

První metoda představuje klasickou sekvenci slidů za pomocí prostředí `frame`. Pokud budeme chtít v jednom slidu zobrazit celý obrázek, vložíme celý jeho zdrojový kód do tohoto prostředí. Ovšem abychom dosáhli postupného odkrývání obsahu, musíme si zdrojový kód obrázku rozdělit do několika částí, podle toho, jak budeme chtít obrázek odkrývat. Tzn. Počet částí obrázku bude roven počtu slidu (počtu prostředí).

Obrázek 7.2: Znázornění obsahu v prvním slidu

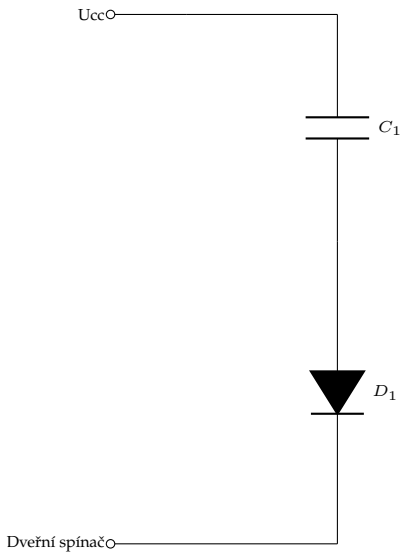


Obrázek 7.3: Znázornění obsahu ve druhém slidu

```
%prvni slide
\begin{frame}
   \begin{circuitikz}
      \draw (0,0) node[left]{Dveřní spínač} to[short, o-](3,0);
   \end{circuitikz}
\end{frame}

%druhy slide
\begin{frame}
   \begin{circuitikz}
      \draw (0,0) node[left]{Dveřní spínač} to[short, o-](3,0);
      \draw (3,4) to[D*, l=$D_1$] (3,0);
   \end{circuitikz}
\end{frame}

..... %dalsí slidy
```

Ze zdrojového kódu je patrné, že jsme vytvořili dva slidy. Obsahem prvního slidu bude obrázek představující vybranou část výsledného obrázku. Druhý slide vykresluje obrázek, který je totožný se slidem předchozím doplněný o další část. Tímto způsobem pokračujeme až do odkrytí výsledného obrázku.

Druhá metoda spočívá v postupném překrývání obsahu, kde se veškerá akce vykoná nad jedním obrázkem umístěný do prostředí `frame`. Není tedy nutné vytvářet další slidy. Překrývání obsahu se provádí za pomocí špičatých závorek <...>. Mezi tyto závorky se uvádí číslo, které znázorňuje na kterém překrytí bude objekt zobrazen. Např. <5> znamená, že se objekt zobrazí pouze na 5 překrytí, zatímco přidáním pomlčky za číslo <5-> způsobí, že se objekt bude zobrazovat i na následujících překrytí. Abychom tuto metodu mohli aplikovat na náš výsledný obrázek, rozdělíme si jej podobně jako v předchozí metodě na více části. Uvnitř každé části za příkazem `draw` umístíme dvojici závorek z požadovaným číslem.



Obrázek 7.4: Postupné překrývání obsahu

```
\begin{frame}
   \begin{circuitikz}
      \draw [draw=white](-2,-1) rectangle (12,8);
      \draw<1-> (0,0) node[left]{Dveřní spínač} to[short, o-](3,0);
      \draw<2-> (3,4) to[D*, l=$D_1$] (3,0);
      \draw<3-> (3,7) to[C, l=$C_1$] (3,4);
      \draw<4-> (3,7) -- (1,7);
      \draw<4-> (1,7) to[short, -o] (0,7) node[left]{Ucc};

      .....

   \end{circuitikz}
\end{frame}
```

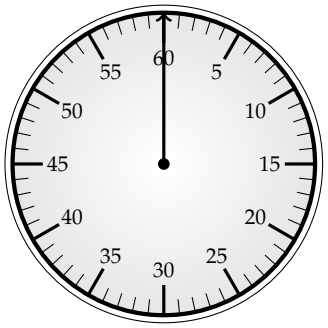
Výsledná prezentace spolu se zdrojovým souborem se nachází na přiloženém CD v adresáři Prezentace (soubory: *Prezentace.pdf* a *Prezentace.tex*).

7.2 Animace

Nyní si ukážeme tvorbu jednoduché a plynulé animace. K tomuto účelu se používá balíček `animate` [11] umístěný do preambule. Pro ukázku jsem vytvořil obrázek znázorňující stopky, jejíž pohyblivou částí bude ručička obíhající po celém obvodu kružnice. Obrázek je složen z několika objektů, kde za zmínku stojí způsob vykreslení větších či menších dílků. K jejich správnému zobrazení použijeme následující příkaz.

```
\draw (úhel : poloměr) --  (úhel : poloměr)
```

Příkaz jasně definuje vykreslení přímky z jednoho bodu do druhého. Body se umístí pod daným úhlem v určené vzdálenosti od středu. Hodnoty těchto úhlů jsou načítány z rozsahu cyklu `foreach`.

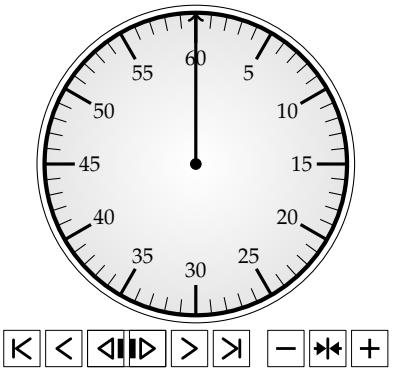


Obrázek 7.5: Ukázkový obrázek znázorňující stopky

```
\begin{tikzpicture}[rotate = 90]
   %vyresleni vsech kruznic
   \draw (0,0) circle (2.1);
   \draw[ultra thick,shading=radial,inner color=white,outer color=gray!20] (0,0) circle (2);
   \draw[fill=black] (0,0) circle (0.07);

   %vykresleni mensich dilku
   \foreach \x in {0, 6,...,360}
      \draw (\x : 1.8) -- (\x : 2);

   %vykresleni vetsich dilku a cislic
   \foreach \x in {5,10,...,60}
   {
      \draw [very thick](\x*6 : 1.6) -- (\x*6 : 2);
      \draw [font=\scriptsize](-\x*6 : 1.4) node {\x};
   }

   %vykresleni rucicky
   \draw[very thick, ->] (0,0) -- (2,0);
\end{tikzpicture}
```

Pokud máme balíček `animate` zaveden, můžeme se pomalu pustit do tvorby animace. Objekty, které chceme rozpohybovat, musíme umístit do prostředí `animateinline`. Ve vlastnostech tohoto prostředí definujeme parametry `loop` a `controls`. První parametr zajistí zacyklení celé animace. Tzn. Pokud do animace manuálně nezasáhneme, bude se vykonávat v nekonečné

smyčce. Druhým parametrem přikládáme k obrázku ovládací prvky, kterými můžeme celou animaci řídit (viz. Obrázek 7.6). Poslední člen, který je třeba doplnit je umístěn ve složených závorkách. Za tento člen se dosazuje číslo určující počet zobrazených snímků za 1 sekundu.



Obrázek 7.6: Statická animace

```
\begin{animateinline}[loop, controls]{6}
   \begin{tikzpicture}[rotate = 90]

      ....

   \end{tikzpicture}
\end{animateinline}
```

Nyní jsme si vytvořili animaci, která nám zatím nic nedělá. Je to způsobeno tím, že se animace skládá pouze z jednoho snímku. Abychom mohli ručičku rozpohybovat, musíme si pro tento účel vytvořit více snímků. Pro tento krok použijeme cyklus `whiledo`, který vykonává příkazy obsažené v těle cyklu tak dlouho, dokud je podmínka umístěna ve složených závorkách splněna. Jakmile je podmínka nepravdivá, cyklus se ukončí. Nejdříve si musíme do preambule nadefinovat dvojici příkazů sloužící k vytvoření pomocné proměnné s přiřazenou hodnotou.

```
\newcounter{prom}
\setcounter{prom}{0}
```

Prvním příkazem vytvoříme proměnnou s jménem `prom`, zatímco druhý příkaz přiřadí této proměnné hodnotu 0. Hodnota této proměnné se v průběhu procházení cyklu `whiledo` navyšuje o 1, použitím příkazu `\stepcounter{prom}`. Každá změna hodnoty bude reprezentovat snímek, který se bude v animaci vyskytovat.

V dalším kroku umístíme obrázek do těla cyklu. Podmínka cyklu nám zajistí, že se vytvoří pouze omezené množství snímku (max. 360 snímků). Rozsah snímků je tedy v rozmezí od 0 do 359. Hodnoty pro vytváření snímků se načítají s proměnné příkazem `\value{prom}`.

```
\begin{animateinline}[loop, controls]{6}
   \whiledo{\value{prom} < 360}
   {
      \begin{tikzpicture}[rotate = 90]

         ....

      \end{tikzpicture}
   }
\end{animateinline}
```

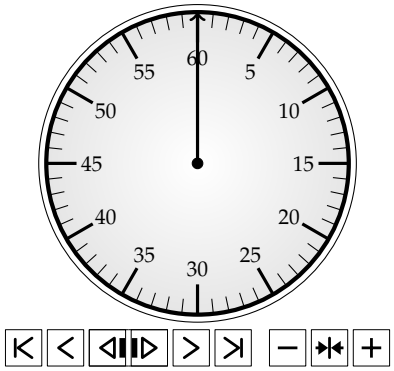
Dalším krokem je doplnění stávajícího kódu o příkaz `ifthenelse`. Je potřeba zavést balíček `ifthen` do preambule. Tento příkaz rozhodne o tom, jak bude kód dále vykonán. V případě kdy je výraz umístěný v podmínce pravdivý, vykoná se první dvojice složených závorek. V případě, že je výraz nepravdivý, vykoná se druhá dvojice. Obsahem první dvojice závorek je příkaz `newframe`, který zajistí vytvoření nového snímku. Tento příkaz bude vykonán pouze tehdy, kdy je hodnota uložená v proměnné menší než 360 (tzn. příkaz se zavolá 360x). Pokud by hodnota proměnné přesáhla tuto hranici, je nutné zajistit, aby se celá animace ukončila. Tohoto kroku docílíme přemístěním příkazu `\end{animateinline}` mezi druhou dvojici složených závorek. Aby se hodnota proměnné `prom` zvyšovala o jedničku, umístíme její příkaz před podmínku `ifthenelse`.

```
\begin{animateinline}[loop, controls]{6}
   \whiledo{\value{prom} < 360}
   {
      \begin{tikzpicture}[rotate = 90]

         ....

      \end{tikzpicture}

      \stepcounter{prom}

      \ifthenelse{\value{prom} < 360}
      {
         \newframe
      }
      {
         \end{animateinline}
      }
   }
\end{animateinline}
```

V tomto okamžiku máme vytvořený obrázek, který se po dobu 360 snímků nijak nemění. Poslední úpravou bude rozpohybování ručičky obíhající po celém obvodu kružnice. Příkaz, který ručičku vykresluje, doplníme o argument `rotate`. Tomuto argumentu následně přiřadíme proměnnou `prom`. Tato úprava nám jednoduše zajistí, otáčení ručičky vždy o jeden stupeň (o hodnotu uloženou v proměnné). Tzn. poloha ručičky v 0°= 1 snímek, 1°= 2 snímek, ... , 359°= 360 snímek.

```
\draw[very thick, ->, rotate = -\value{prom}] (0,0) -- (2,0);
```

Obrázek 7.7 zobrazuje výslednou animaci.



Obrázek 7.7: Výsledná animace

```
%zacatek animace
\begin{animateinline}[loop, controls]{6}

%cyklus definujíci pocet snimku
\whiledo{\value{prom} < 360}
{
  \begin{tikzpicture}[rotate = 90]
    %vyresleni vsech kruznic
    \draw (0,0) circle (2.1);
    \draw[ultra thick,shading=radial,inner color=white,
        outer color=gray!20] (0,0) circle (2);
    \draw[fill=black] (0,0) circle (0.07);
    %vykresleni mensich dilku
    \foreach \x in {0, 6,...,360}
       \draw (\x : 1.8) -- (\x : 2);
    %vykresleni vetsich dilku a cislic
    \foreach \x in {5,10,...,60}
    {
       \draw [very thick](\x*6 : 1.6) -- (\x*6 : 2);
       \draw [font=\scriptsize](-\x*6:1.4) node {\x};
    }
    %vykresleni rucicky
    \draw[very thick, ->, rotate = -\value{prom}] (0,0) -- (2,0);
  \end{tikzpicture}
  %inkrementace promenne (zvyseni o 1)
  \stepcounter{prom}
  %podminka ifelse
  \ifthenelse{\value{prom} < 360}
  {
     \newframe   %vytvoreni noveho snimku
  }
  {
     \end{animateinline}   %ukonceni animace
  }
}
```

Animaci můžeme taktéž vykreslit do prezentace. Prezentace spolu se zdrojovým souborem se nachází na přiloženém CD v adresáři Prezentace (soubory: *PrezentaceAnimace.pdf* a *PrezentaceAnimace.tex*).

8 Tvorba vlastních tvarů

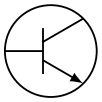
Do této kapitoly jsme si ukázali sazbu obrázků, složených z menších námi vytvořených objektů za pomocí příkazů `draw`, `node`, apod. Taktéž jsme si ukázali jakým způsobem můžeme využívat a sázet složitější tvary, které jsme získali zavedením různých balíčků do dokumentu. Tyto tvary byly vytvořeny za účelem zjednodušit práci uživatelům při sazbě obrázků. Uživatelé se nemusí příliš zajímat o to, z jakých objektů je tvar vytvořen, musí znát pouze jeho specifický název a správnou syntaxi pro jeho užití. Ovšem může nastat situace, kdy nám sada prvků (tvarů) obsažených v balíčků nevyhovuje popř. máme jinou představu o tom, jak má daný tvar vypadat. Takže nám nezbývá mnoho možností, buď se poohlédneme po jiném balíčku, který by splňoval naše požadavky nebo si vytvoříme vlastní tvar.

Pokud se rozhodneme pro tvorbu vlastního tvaru, můžeme k tomuto účelu použít základní vrstvu PGF. I když je tvorba tvarů na této vrstvě poměrně složitá a vyžaduje více znalostí, poskytuje nám mnoho příkazů, kterými můžeme vytvářet velice propracované grafické ilustrace. My si v této kapitole na této vrstvě ukážeme tvorbu velice jednoduchých vlastních tvarů, které si následně vykreslíme za pomoci vrstvy Ti*k*Z.

8.1 Tranzistory PNP a NPN s pevnou velikostí

V kapitole 5 jsme si vykreslili několik ukázkových obrázků znázorňující elektronická schémata. Pro jejich tvorbu jsme použili balíček `circuitikz`, jehož obsahem je sada nejpoužívanějších elektronických značek a logických prvků. Pro ukázku si upravíme jeden z obrázků a do něj pak následně vysázíme vlastní tvar.

Tvar, který si vytvoříme bude schématická značka tranzistoru NPN s pevně danou velikostí.



Obrázek 8.1: Schématická značka tranzistoru NPN

K vytvoření vlastního tvaru použijeme příkaz `\pgfdeclareshape{název tvaru}{...}`. První dvojice složených závorek slouží pro pojmenování tvaru. Kdykoliv budeme chtít tvar vyobrazit, použijeme právě toto jméno. Do složených závorek druhé dvojice se pak vkládají jednotlivé příkazy, které definují, jak bude tvar vypadat, popř. jaké bude mít vlastnosti.

```
\pgfdeclareshape{transistorNPN}
{
   ...
}
```

Jakmile máme příkaz pro deklaraci tvaru zaveden, můžeme do jeho prostředí vkládat speciální příkazy. Jelikož ohraničení tranzistoru je tvořené kružnicí, bylo by vhodné, kdyby zdědil některé její vlastnosti. Např. umístění kotev (připojovacích bodů), ke kterým mohu kdykoliv připojit text. Nejdříve si do bloku zavedeme příkaz `\inheritsavedanchors`. Tento příkaz

umožňuje zdědit polohu uložených kotev z tvaru kružnice. Abychom je ale blíže specifikovali, zavedeme pro každou konkrétní kotvu příkaz `\inheritanchor`.

```
\pgfdeclareshape{transistorNPN}
{
   \inheritsavedanchors[from=circle]
   \inheritanchor[from=circle]{center}
   \inheritanchor[from=circle]{east}
   \inheritanchor[from=circle]{west}
   \inheritanchor[from=circle]{north}
   \inheritanchor[from=circle]{south}
}
```

Tímto způsobem se nám podařilo zdědit některé konkrétní kotvy z tvaru kružnice. Bylo by vhodné definovat si vlastní kotvy, ke kterým se chceme z venku připojovat. Jelikož tranzistor obsahuje tři elektrody (B-báze, E-emitor, C-kolektor), je zapotřebí pro každou elektrodu vytvořit vlastní kotvu.

K tomuto kroku použijeme příkaz `\anchor{název kotvy}{poloha kotvy}`. Je nezbytné si kotvu pojmenovat, neboť právě toto jméno nám umožní se odkázat na přesnou polohu kotvy. Pro definování polohy použijeme příkazy `\pgfpoint{X}{Y}`, který umístí bod v souřadnicích (X,Y) a `\pgfpointpolar{úhel}{poloměr}`, který nám umístí bod na kružnici o námi definovaném úhlu. Pokud neuvedeme jinou jednotku, budou všechny hodnoty ve výchozím stavu nastaveny v bodech *(pt)*.

```
\pgfdeclareshape{transistorNPN}
{
   \inheritsavedanchors[from=circle]
   \inheritanchor[from=circle]{center}
   \inheritanchor[from=circle]{east}
   \inheritanchor[from=circle]{west}
   \inheritanchor[from=circle]{north}
   \inheritanchor[from=circle]{south}

   \anchor{B}{\pgfpoint{-10.5}{0}}
   \anchor{C}{\pgfpointpolar{45}{10.5}}
   \anchor{E}{\pgfpointpolar{315}{10.5}}
}
```

V tomto okamžiku máme nastaveny všechny potřebné vlastnosti pro daný tvar. Zbývá nám do něj zakreslit cesty, které zajistí, jak bude tvar vypadat. Všechny příkazy sloužící pro vykreslování cest, umístíme do prostředí příkazu `\backgroundpath{...}`, který se chová jako pozadí tvaru.

```
\pgfdeclareshape{transistorNPN}
{
   \inheritsavedanchors[from=circle]
   \inheritanchor[from=circle]{center}
   \inheritanchor[from=circle]{east}
   \inheritanchor[from=circle]{west}
   \inheritanchor[from=circle]{north}
   \inheritanchor[from=circle]{south}

   \anchor{B}{\pgfpoint{-10.5}{0}}
   \anchor{C}{\pgfpointpolar{45}{10.5}}
   \anchor{E}{\pgfpointpolar{315}{10.5}}

   \backgroundpath
   {
```

```
            ... %prikazy pro vykresleni cesty
        }
    }
```

První příkaz, který použijeme je `\pgfpathcircle{(poloha středu)}{poloměr}`. Slouží pro vykreslení kružnice se středem (X,Y) o zvoleném poloměru.

Obrázek 8.2: Vykreslení kružnice na pozadí tvaru

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        \pgfpathcircle{\pgfpoint{0}{0}}{10.5}
    }
}
```

Jakmile jsme kružnici vykreslili, dalším objektem bude vykreslení středové přímky. Pro vykreslení přímky se v PGF používá dvojice příkazů:

`\pgfpathmoveto{poloha bodu}`
`\pgfpathlineto{poloha bodu}`

První příkaz určí polohu počátečního bodu, zatímco druhý příkaz určí polohu následujícího bodu, který zároveň mezi těmito body vykreslí cestu.

Obrázek 8.3: Vykreslení vnitřní cesty na pozadí tvaru

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        ... %prikazy vykreslujici kruznici

        %vnitrni cara
        \pgfpathmoveto{\pgfpoint{-2.5}{5}}
        \pgfpathlineto{\pgfpoint{-2.5}{-5}}
    }
}
```

Dalším krokem bude vykreslení cesty reprezentující bázi tranzistoru.

Obrázek 8.4: Vykreslení cesty reprezentující bázi na pozadí tvaru

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        ... %prikazy vykreslujici kruznici, vnitrni caru

        %baze
        \pgfpathmoveto{\pgfpoint{-2.5}{0}}
        \pgfpathlineto{\pgfpoint{-10.5}{0}}
    }
}
```

Pro vykreslení cesty znázorňující elektrody kolektor a emitor použijeme stejnou metodu, ovšem poloha následujícího bodu se musí rovnat poloze definované v našich vytvořených kotvách (C,E).

Obrázek 8.5: Vykreslení cesty reprezentující kolektor na pozadí tvaru

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        ... %prikazy vykreslujici kruznici, vnitrni caru, bazi

        % kolektor
        \pgfpathmoveto{\pgfpoint{-2.5}{2}}
        \pgfpathlineto{\pgfpointpolar{45}{10.5}}
    }
}
```

Nyní nám zbývá dokreslit cestu reprezentující emitor. Emitor se vždy vykresluje se šipkou, tudíž musíme kód doplnit o následující příkazy:

```
\pgfsetarrowsend{latex}
\pgfusepathqstroke
\pgfusepath{stroke}
```

První příkaz umožní vykreslit šipku (latex) na konec cesty. Druhý příkaz je nutné definovat, neboť nám zaručí vykreslení šipky. Poslední příkaz slouží už k samotnému vykreslení cesty s šipkou.

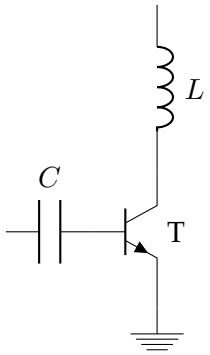
Obrázek 8.6: Vykreslení cesty reprezentující emitor na pozadí tvaru

```
\pgfdeclareshape{transistorNPN}
{
   \inheritsavedanchors[from=circle]
   \inheritanchor[from=circle]{center}
   \inheritanchor[from=circle]{east}
   \inheritanchor[from=circle]{west}
   \inheritanchor[from=circle]{north}
   \inheritanchor[from=circle]{south}

   \anchor{B}{\pgfpoint{-10.5}{0}}
   \anchor{C}{\pgfpointpolar{45}{10.5}}
   \anchor{E}{\pgfpointpolar{315}{10.5}}

   \backgroundpath
   {
      % kruznice
      \pgfpathcircle{\pgfpoint{0}{0}}{10.5}
      % vnitrni cara
      \pgfpathmoveto{\pgfpoint{-2.5}{5}}
      \pgfpathlineto{\pgfpoint{-2.5}{-5}}
      % kolektor
      \pgfpathmoveto{\pgfpoint{-2.5}{2}}
      \pgfpathlineto{\pgfpointpolar{45}{10.5}}
      %baze
      \pgfpathmoveto{\pgfpoint{-2.5}{0}}
      \pgfpathlineto{\pgfpoint{-10.5}{0}}
      % emitor + sipka
      \pgfusepathqstroke
      \pgfsetarrowsend{latex}
      \pgfpathmoveto{\pgfpoint{-2.5}{-2}}
      \pgfpathlineto{\pgfpointpolar{315}{10.5}}
      \pgfusepath{stroke}
   }
}
```

Toto je náš výsledný kód, který je ještě nutné umístit do preambule. V případě, že bychom chtěli vytvořit tranzistor PNP, je zapotřebí v kódu zaměnit příkaz `\pgfsetarrowsend{latex}` za příkaz `\pgfsetarrowsstart{latex}`. První příkaz umístí šipku na konec cesty, druhý příkaz jí naopak umístí na začátek. Tvar reprezentující tranzistor PNP si pojmenujeme jako „transistorPNP“.

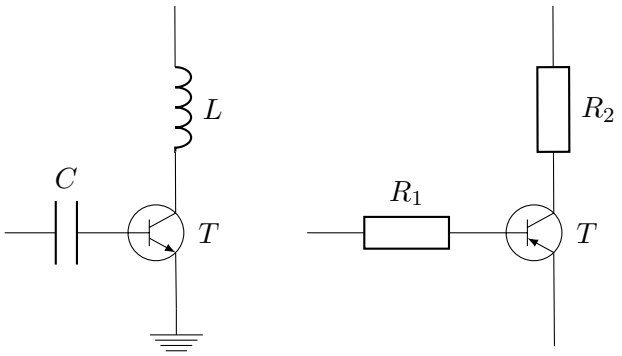
V následujících příkladech si ukážeme, jak můžeme vlastní vytvořené tvary vykreslit vrstvou Ti*k*Z.

Upravíme si tedy obrázek 8.7, jehož zdrojový kód nalezneme v kapitole 5 (obrázek 5.7).



Obrázek 8.7: Ukázkový příklad pro vykreslení tranzistoru balíkem `circuitikz`

Obrázek 8.8 vykresluje dva elektronická schémata. V prvním schématu nahradíme tranzistor z předchozího obrázku vlastním tvarem, zatímco druhé schéma slouží pro ukázku vykreslení tranzistoru PNP. Za pomoci námi vytvořených kotev (B, E, C) můžeme k tranzistorům připojovat jiné tvary.



Obrázek 8.8: Vykreslení schéma s vlastním tvarem (tranzistorNPN/PNP) s pevnou velikostí

```
\begin{circuitikz}
   \begin{scope}
     \draw (0,0) node[transistorNPN] (Tnpn) {} node [xshift = .7cm]{$T$};
     \draw (-2,0) to[C, l=$C$] (Tnpn.B);
     \draw (0.25,3) to[american inductor, l=$L$] (Tnpn.C);
     \draw (0.27,-1) node[ground]{} -- (Tnpn.E);
   \end{scope}

   \begin{scope}[xshift=5cm]
     \draw (0,0) node[transistorPNP] (Tpnp) {} node [xshift = .7cm]{$T$};
     \draw (-3,0) to[generic, l=$R_1$] (Tpnp.B);
     \draw (0.25,3) to[generic, l=$R_2$] (Tpnp.C);
     \draw (0.27,-1.5) -- (Tpnp.E);
   \end{scope}
\end{circuitikz}
```

8.2 Tranzistor NPN - změna velikosti tvaru

Do této části jsme si vytvořili tvar, který jsme požadovali. Avšak definováním hodnot přímo v kódu způsobí, že nebude možné změnit velikost tvaru. To je ovšem docela důležitá věc, protože ne vždy potřebujeme tvar ve stejné velikosti, v jaké jsme jej vytvořili. Upravíme si náš současný kód tak, abychom mohli podle potřeby kdykoliv změnit celkovou velikost tvaru. Velikost tvaru je řízena argumentem `minimum size` uloženým ve vlastnostech příkazu sloužící pro vykreslení tvaru.

První změna v kódu bude věnována vlastním kotvám. Chceme, aby se při každé změně hodnoty argumentu `minimum size` upravila i přesná poloha kotev. Toho docílíme zavedením příkazu `\radius`. Jestliže k argumentu `minimum size` přiřadíme hodnotu 1 cm, bude automaticky příkaz `\radius` nabývat hodnoty 0.5 cm, což se nám hodí, neboť se kotva umístí přesně na kružnici.

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy pro definici ulozenych kotev zustavaji beze zmen

    \anchor{B}{\pgfpoint{-\radius}{0}}
    \anchor{C}{\pgfpointpolar{45}{\radius}}
    \anchor{E}{\pgfpointpolar{315}{\radius}}
}
```

V tomto okamžiku jsme vyřešili problém u námi vytvořených kotev. Nyní tutéž problematiku musíme aplikovat na cesty uvnitř příkazu `\backgroundpath`.

První příkaz, který definujeme se nazývá `\pgfsetlinewidth` a slouží k určení tloušťky čáry (cesty). V závislosti na změně hodnoty uložené v příkazu `\radius` vynásobené pomocnou hodnotou, dojde k upravě tloušťky všech cest uložených v příkazu `\backgroundpath`.

Příkaz pro vykreslení kružnice doplníme o `\radius` a `\pgfpointorigin`. Podobně jako u vytvořených kotev musíme nastavit poloměr kružnice.
Druhý příkaz reprezentuje `\pgfpoint{0pt}{0pt}`.

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        %nastaveni tloustky car
        \pgfsetlinewidth{0.04 * \radius}
        % kruznice
        \pgfpathcircle{\pgfpointorigin}{\radius}
    }
}
```

Abychom mohli upravit i zbylé cesty, je nutné do kódu zavést dočasné proměnné `\pgf@x` a `\pgf@y`. Slouží jako vnitřní registry používané pro získání hodnot v souřadnicích X a Y. Např. Pokud v kódu použijeme příkaz `\pgfpoint{2pt}{5pt}`, automaticky se hodnoty uloží do dočasných proměnných (`\pgf@x = 2pt` a `\pgf@y = 5pt`). Pokud bychom v dalším kroku použili jiný příkaz pracující se souřadnicemi X a Y, hodnoty uložené v dočasných proměnných se automaticky přepíšou novými hodnotami.

Toto je určitě užitečná vlastnost, ovšem bylo by dobré si definovat proměnné, jejíž hodnoty se nebudou přepisovat. K tomuto účelu máme k dispozici registry `\pgf@xa`, `\pgf@ya`, `\pgf@xb`,

`\pgf@yb, ...` Nad těmito registry budeme následně provádět další operace. Jednou z operací, kterou budeme v kódu často používat je příkaz `\advance` zabývající se početními operacemi. Tento příkaz budeme využívat pro napozicování cest.

```
\advance\pgf@xb by 0.17\pgf@x
```

Pomoci tohoto příkazu, provedeme následující operaci, kdy vynásobíme hodnotu 0,17 z hodnotou uloženou v registru `\pgf@x`, ta se následně přičte k hodnotě, která je uložena v registru `\pgf@xb` (zatím není nikde použita, proto nabývá hodnoty 0). Získanou hodnotu uloženou v registru `\pgf@xb` poté použijeme pro napozicování bodu na X souřadnici. Hodnotu uloženou v registru `\pgf@x` jsme získali z příkazu `\pgfpathcircle` načtením velikosti poloměru. Podobným způsobem si vypočítáme hodnotu pro registr `\pgf@yb` k napozicování bodu Y souřadnice. Registry následně použijeme pro vykreslení cesty reprezentující vnitřní čáru.

```
\pgfdeclareshape{transistorNPN}
{
    ... %prikazy definujici kotvy

    \backgroundpath
    {
        %nastaveni tloustky car
        \pgfsetlinewidth{0.04 * \radius}
        % kruznice
        \pgfpathcircle{\pgfpointorigin}{\radius}

            \advance\pgf@xb by 0.17\pgf@x
            \advance\pgf@yb by 0.5\pgf@x
        % inner line
        \pgfpathmoveto{\pgfpoint{-\pgf@xb}{\pgf@yb}}
        \pgfpathlineto{\pgfpoint{-\pgf@xb}{-\pgf@yb}}
    }
}
```

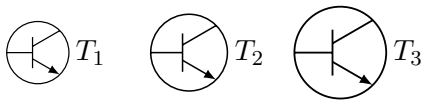
Tímto způsobem můžeme napozicovat body pro všechny zbývající cesty. Následující kód je doplněn o všechny cesty reprezentující bázi, kolektor a emitor.

V tomto okamžiku jsme si vytvořili tvar s proměnlivou velikostí. Ovšem je zapotřebí učinit následující kroky, které jsou nutné pro práci s tímto tvarem.

Aby nám tvar fungoval, musíme příkaz `\pgfdeclareshape { ... }` umístit mezi dvojici příkazů `\makeatletter` a `\makeatother`. Umístění celého bloku mezi tyto příkazy způsobí, že můžeme ve svém kódu používat makra, které obsahují ve svém názvu @. Celý tento blok pak umístíme do preambule.

```
\makeatletter
   \pgfdeclareshape{transistorNPN}
   {
      \inheritsavedanchors[from=circle]
      \inheritanchor[from=circle]{center}
      \inheritanchor[from=circle]{east}
      \inheritanchor[from=circle]{west}
      \inheritanchor[from=circle]{north}
      \inheritanchor[from=circle]{south}

      \anchor{B}{\pgfpoint{-\radius}{0}}
      \anchor{C}{\pgfpointpolar{45}{\radius}}
      \anchor{E}{\pgfpointpolar{315}{\radius}}
      \backgroundpath
      {
         %nastaveni tloustky car
         \pgfsetlinewidth{0.04 * \radius}
         % kruznice
         \pgfpathcircle{\pgfpointorigin}{\radius}
         % inner line
            \advance\pgf@xb by 0.17\pgf@x
            \advance\pgf@yb by 0.5\pgf@x
         \pgfpathmoveto{\pgfpoint{-\pgf@xb}{\pgf@yb}}
         \pgfpathlineto{\pgfpoint{-\pgf@xb}{-\pgf@yb}}
         % base
            \pgf@xa = \radius
         \pgfpathmoveto{\pgfpoint{-\pgf@xb}{0}}
         \pgfpathlineto{\pgfpoint{-\pgf@xa}{0}}
         % collector
            \advance\pgf@yb by -1.4\pgf@yb
         \pgfpathmoveto{\pgfpoint{-\pgf@xb}{\pgf@yb}}
         \pgfpathlineto{\pgfpointpolar{45}{\pgf@xa}}
         % emitter (with arrow)
            \advance\pgf@yb by -2\pgf@yb
         \pgfusepathqstroke
         \pgfsetarrowsend{latex}
         \pgfpathmoveto{\pgfpoint{-\pgf@xb}{\pgf@yb}}
         \pgfpathlineto{\pgfpointpolar{315}{\pgf@xa}}
         \pgfusepath{stroke}
         \pgfsetarrowsend{}
      }
   }
\makeatother
```

Na následujícím obrázku můžeme zpozorovat vykreslení tvarů o různých velikostech. Velikost každého tvaru je závislá na hodnotě, která je uvedena u argumentu `minimum size` ve vlastnostech každého tvaru.

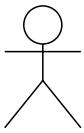


Obrázek 8.9: Vykreslení tvarů (tranzistorNPN) s proměnlivou délkou

```
\begin{tikzpicture}
   \draw (0,0) node[transistorNPN, minimum size=23pt] (T1){} node [xshift = .7cm]{$T_1$};
   \draw (2,0) node[transistorNPN, minimum size=1cm] (T2){} node [xshift = .8cm]{$T_2$};
   \draw (4,0) node[transistorNPN, minimum size=1.2cm] (T3){} node [xshift = .9cm]{$T_3$};
\end{tikzpicture}
```

8.3 UML - Aktér

K tvorbě vývojových diagramů existuje spousta balíčků, které obsahují nepřeberné množství nejrůznějších tvarů, kterými můžeme popsat jakýkoliv problém řešící určitou situaci. My si tyto balíčky popisovat nebudeme a rovnou si ukážeme, jak pro diagram případu užití vytvořit tvar reprezentující aktéra (Actor).



Obrázek 8.10: Diagram případu užití - Aktér

Prvním krokem bude deklarace nového tvaru, který si pojmenujeme jako „Actor“.

```
\pgfdeclareshape{Actor}
{
   ...
}
```

Jelikož budeme k našemu tvaru připojovat další objekty, bylo by vhodné zdědit vlastnosti a uložené kotvy z jiného tvaru. Pro náš případ zvolíme tvar `rectangle`. Je teď pouze na nás, jaké kotvy specifikujeme. Taktéž si zavedeme příkaz `\inheritanchorborder`, který nám zajistí, že se objekty budou moci připojit pouze k okraji tvaru. Náš tvar obsahuje několik cest včetně kružnice, proto je nezbytné si v kódu zároveň definovat příkaz pro dědění uložených kotev z tvaru kružnice.

```
\pgfdeclareshape{Actor}
{
   \inheritsavedanchors[from=rectangle]
   \inheritanchor[from=rectangle]{center}
   \inheritanchor[from=rectangle]{north}
   \inheritanchor[from=rectangle]{south}
   \inheritanchor[from=rectangle]{east}
   \inheritanchor[from=rectangle]{west}
   \inheritanchor[from=rectangle]{north east}
   \inheritanchor[from=rectangle]{north west}
   \inheritanchor[from=rectangle]{south east}
   \inheritanchor[from=rectangle]{south west}
   \inheritanchorborder[from=rectangle]

   \inheritsavedanchors[from=circle]
}
```

Nyní se pustíme do samotného vykreslování cest. Podobně jako u tranzistoru je určitě vhodné, definovat příkaz pro změnu tloušťky čar. Abychom mohli vypočítat tloušťku čáry, musíme nejdříve načíst hodnotu uloženou v argumentu `minimum size`.
Jelikož dědíme z tvaru `rectangle`, máme v tomto případě k dispozici příkazy `\northeast` a `\southwest`, které jednoduše reprezentují body pravého horního rohu a levého dolního rohu tvaru `rectangle`.
Např. pokud zvolíme `minimum size = 2cm`, bude tvar zobrazen jako čtverec o stranách max. 2 cm. Na osách XY se tvar vykreslí od středu, kde hodnoty X a Y = 1 pro `\northeast`, kdežto X a Y pro `\southwest` nabývají hodnoty -1.

Pro výpočet tloušťky čar využijeme příkaz `\northeast`. Definováním tohoto příkazu získáme hodnotu, kterou si automaticky převezme dočasná proměnná `\pgf@x`.
Hodnotu této proměnné si vzápětí uložíme do registru `\pgf@xa`, který následně použijeme v příkazu `\pgfsetlinewidth`.

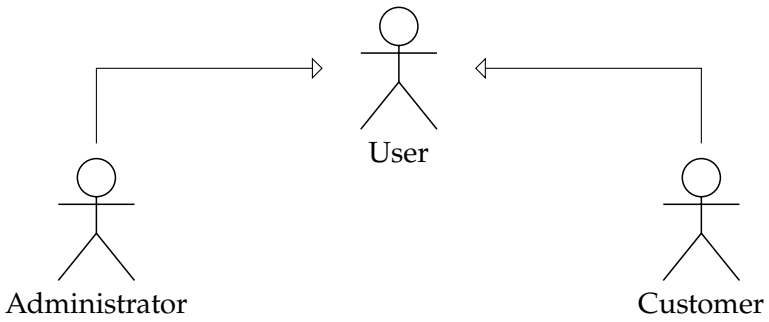
```
\pgfdeclareshape{Actor}
{
   \inheritsavedanchors[from=rectangle]
   \inheritanchor[from=rectangle]{center}
   \inheritanchor[from=rectangle]{north}
   \inheritanchor[from=rectangle]{south}
   \inheritanchor[from=rectangle]{east}
   \inheritanchor[from=rectangle]{west}
   \inheritanchor[from=rectangle]{north east}
   \inheritanchor[from=rectangle]{north west}
   \inheritanchor[from=rectangle]{south east}
   \inheritanchor[from=rectangle]{south west}
   \inheritanchorborder[from=rectangle]
   \inheritsavedanchors[from=circle]
   \backgroundpath
   {
      %nastaveni tloustky car
      \pgf@process{\northeast}
         \pgf@xa = \pgf@x
         \pgfsetlinewidth{0.02 * \pgf@xa}
   }
}
```

Za pomocí příkazů `\northeast` a `\southwest` získáme hodnoty potřebné pro výpočet ostatních cest. Podobně jako u tranzistoru si budeme muset pro správné napozicování bodů pomoci příkazem `\advance`. Následný kód vykresluje všechny potřebné cesty reprezentující části postavičky. V případě, že bychom chtěli v obrázku vytvořit více stejných tvarů, bylo by vhodné, kdybychom mohli ke každému z nich přiřadit popisek. Tento problém můžeme vyřešit deklarováním vlastní kotvy, kterou si pojmenujeme jako „text“.

```
\makeatletter
  \pgfdeclareshape{Actor}
  {
     \inheritsavedanchors[from=rectangle]
     \inheritanchor[from=rectangle]{center}
     \inheritanchor[from=rectangle]{north}
     \inheritanchor[from=rectangle]{south}
     \inheritanchor[from=rectangle]{east}
     \inheritanchor[from=rectangle]{west}
     \inheritanchor[from=rectangle]{north east}
     \inheritanchor[from=rectangle]{north west}
     \inheritanchor[from=rectangle]{south east}
     \inheritanchor[from=rectangle]{south west}
     \inheritanchorborder[from=rectangle]
     \inheritsavedanchors[from=circle]
     \anchor{text}
     {
       \southwest
        \pgf@xa = \pgf@x  \pgf@ya = \pgf@y
        \advance\pgf@xa by -\pgf@xa
        \advance\pgf@ya by 0.1\pgf@ya
       \pgfpoint{\pgf@xa}{\pgf@ya}
     }
     \backgroundpath
```

```
        {
            %nastaveni tloustky car
            \pgf@process{\northeast}
            \pgf@xa = \pgf@x
            \pgfsetlinewidth{0.02 * \pgf@xa}
            %TELO
            \pgf@process{\northeast}
               \pgf@xa = \pgf@x  \pgf@ya = \pgf@y
               \pgf@ya = 0.3\pgf@ya
               \pgf@yb = -0.6\pgf@ya
               \pgfpathmoveto{\pgfpoint{0}{\pgf@ya}}
               \pgfpathlineto{\pgfpoint{0}{\pgf@yb}}
            %NOHY
            \pgf@process{\southwest}
               \pgf@xc = \pgf@x  \pgf@yc = \pgf@y
               \advance\pgf@xc by -0.5\pgf@xc
               \advance\pgf@yc by -0.2\pgf@yc
               \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}
               \pgfpathmoveto{\pgfpoint{0}{\pgf@yb}}
               \advance\pgf@xc by -2\pgf@xc
               \pgfpathlineto{\pgfpoint{\pgf@xc}{\pgf@yc}}
            % RUCE
            \pgf@process{\northeast}
               \pgf@xa = \pgf@x  \pgf@ya = \pgf@y
               \pgf@xb = \pgf@x
               \advance\pgf@xa by -0.5\pgf@xa
               \advance\pgf@xb by -1.5\pgf@xb
               \advance\pgf@ya by -0.8\pgf@ya
               \pgfpathmoveto{\pgfqpoint{\pgf@xa}{\pgf@ya}}
               \pgfpathlineto{\pgfqpoint{\pgf@xb}{\pgf@ya}}
            %HLAVA
            \pgf@process{\northeast}
               \pgf@xa = \pgf@x  \pgf@ya = \pgf@y
               \advance\pgf@xa by -1\pgf@x
               \advance\pgf@ya by -0.45\pgf@y
               \pgfpathcircle{\pgfpoint{\pgf@xa}{\pgf@ya}}{(0.25 * \radius)}
            % vykresli cesty
            \pgfusepath{draw}
        }
     }
   \makeatother
```

Následující obrázek vykresluje tři aktéry mezi kterými je použit vztah generalizace. Ke každému tvaru je za pomocí kotvy `text` připojený popisek.



Obrázek 8.11: Vykreslení tvarů (Aktér)

```
\begin{tikzpicture}[general-/.style={{triangle 90}-, fill=white}]
   \draw (4,2) node[Actor,minimum size=2cm] (user) {};
   \node at (user.text) {User};

   \draw (0,0) node[Actor, minimum size=2cm] (admin) {};
   \node at (admin.text) {Administrator};

   \draw (8,0) node[Actor, minimum size=2cm] (customer) {};
   \node at (customer.text) {Customer};

   \draw [general-] (user.west) -| (admin.north);
   \draw [general-] (user.east) -| (customer.north);
\end{tikzpicture}
```

9 Závěr

Cílem bakalářské práce bylo popsat práci s balíčkem Ti*k*Z, který je určený pro sazbu grafických ilustrací v systému LaTeX. Práce se měla zaměřit na typické příklady obrázků v oblasti elektrotechniky a informatiky.
Před začátkem realizace bakalářské práce jsem s balíčkem neměl žádnou zkušenost. Po prvním prostudování jsem nabyl dojmu, že se jedná svými funkcemi o velice rozsáhlý balík umožňující vytvářet velice propracovanou grafiku. Z tohoto důvodu jsem se rozhodl práci pojmout jako jednoduchou příručku pro začínající uživatele.
Jednotlivé kapitoly jsem seřadil podle složitosti, od naprostých začátků, kde popisuji základní práci s balíčkem Ti*k*Z až po složitější tvorbu obrázků. S přibývajícími příkazy se každá kapitola věnuje určitému tématu, v němž jsou popsány postupy pro tvorbu požadovaných obrázků. Ke každé grafické ilustraci je zobrazen zdrojový kód, který se taktéž nachází na přiloženém CD.
Ačkoliv má tento balík velmi dobře zpracovaný a rozsáhlý manuál, častokrát jsem byl nucen porozhlédnout se po jiném řešení určité situace. Spoustu velmi užitečný rad jsem čerpal z ukázkových příkladů [9] jiných zkušenějších uživatelů.
Práce s balíčkem Ti*k*Z mi ukázala jiný způsob tvorby grafických ilustrací pro technické dokumentace. Svojí jednoduchou a přehlednou syntaxi jsem dokázal v krátkém čase vytvářet požadované obrázky. Jeho velkou přednost jsem také zpozoroval při tvorbě animací a prezentací s použitím třídy `Beamer`.
Ačkoliv je možné za pomocí vrstvy Ti*k*Z vytvářet kvalitní grafiku, ne vždy si vystačíme pouze s jeho příkazy. U složitějších ilustrací popř. deklarací vlastních tvarů je nutné sáhnout po příkazech nižší (základní) vrstvy PGF. Z tohoto důvodu by bylo určitě vhodné podrobněji seznámit uživatele i se základní vrstvou PGF na které je vrstva Ti*k*Z postavena.

Tomáš Pavlorek

10 Reference

[1] Tantau, Till. *The TikZ and PGF Packages*, 2010.
http://www.texample.net/tikz/

[2] Kopka, Helmut a Daly, W., Patrick. *LaTeXKompletní průvodce*, 2004.

[3] Rybička, Jiří. *LaTeXpro začátečníky 3. vyd.*, Brno: 2003.

[4] Sturm, F., Thomas. *The csvsimple package*, 2012.
http://mirrors.ibiblio.org/CTAN/macros/latex/contrib/csvsimple/

[5] Redaelli, A., Massimo. *The circuiTikZ manual*, 2012.
http://get-software.net/graphics/pgf/contrib/circuitikz/

[6] Lamport, Leslie, *LaTeX: a document preparation system: user's guide and reference manual*, New York: Addison-Wesley Pub. Co., 1994.

[7] Tantau, Till. *The Beamer class*, 2007.
http://www.tex.ac.uk/tex-archive/macros/latex/contrib/beamer/

[8] Vavříček, Jan. *LaTeX - VavříčekWiki [online]*, 2013.
http://vavricek.cs.vsb.cz/index.php/LaTeX

[9] *TikZ and PGF examples [online]*, 2013.
http://www.texample.net/tikz/examples/

[10] *CSTUG dokumenty a manuály [online]*, 2013.
http://www.cstug.cz/documentation/

[11] Grahn, Alexander. *The Animate Package*, 2012.
http://www.ctan.org/tex-archive/macros/latex/contrib/animate

[12] *Jak na LaTeX [online]*, 2013.
http://www.root.cz/serialy/jak-na-latex/